Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering

Real-time teleoperation of a robot arm for manipulating self-localization in human participants

Oleg Baryshnikov

Supervisor: Doc. Mgr. Matěj Hoffmann, Ph.D.
Supervisor–specialist: Sergiu Tcaci Popescu, Ph.D.
Field of study: Open Informatics
Subfield: Software Engineering
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Baryshnikov Oleg**	Personal ID number:	**507458**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Software Engineering**		

II. Master’s thesis details

Master’s thesis title in English:

Real-time teleoperation of a robot arm for manipulating self-localization in human participants

Master’s thesis title in Czech:

Teleoperace robotické paže v reálném čase pro manipulaci sebelokalizace u lidských účastníků

Guidelines:

The mechanisms of how humans localize touch on their bodies are not fully understood. To increase understanding, specific manipulations of tactile localization are needed. One possibility is to exploit self-contact when the human is sliding over its skin surface, but insert a robot arm in the middle [CAT22]. However, state-of-the-art studies lack the ecological conditions of free movement. A motion capture system (Qualisys) coupled with a teleoperated robotic manipulator equipped with an artificial finger will bridge the gap and allow more freedom to study the influence of proprioception vs. touch during self-touch by changing forward kinematics parameters [PAT12].
This thesis follows up on [ROJ23] and extends it into movements on a plane (2D space) with the possibility of different manipulations of the spatial transformation (human arm to robot arm). The thesis will also provide a mature GUI interface for running and configuring the experiments and data logging.
The commission for Ethics in Research at the Czech Technical University in Prague has approved the experiments and the informed consent form. Experiments with participants will be conducted and evaluated during work on the thesis.
Tasks:
1. Familiarize yourself with the Kinova Gen3 robotic platform, Qualisys motion tracking system, psychological experiment procedures [CAT22], and previous implementation [ROJ23].
2. Extend the existing implementation (and reimplement when necessary) by adding an interface to configure the experiments (for non-programmers), a familiarization phase for participants, and logging of the data for evaluation of the experiments. Extend the existing GUI where necessary. Rework existing software architecture and use suitable architecture and design patterns if needed.
3. Analyze critical parts of the program and cover it by Unit-tests. Pay specific attention to the safety of participants - see Risk assessment in [ROJ23]. It has to be guaranteed that the robot arm does not leave the defined workspace, exceed set velocity limits, or contact forces.
4. Together with the supervisors, run experiments with participants (1D version - participants and robot moving on a line).
5. Extend the setup to a plane (2D): Participant moves its arm on a plane which is mapped to the motion of the robot arm (subject to manipulations), which then touches the participant’s other arm.
6. Provide comprehensive documentation.

Bibliography / sources:

[CAT22] Cataldo, A., Dupin, L., Dempsey-Jones, H., Gomi, H., & Haggard, P. (2022). Interplay of tactile and motor information in constructing spatial self-perception. Current Biology, 32(6), 1301-1309.
[PAT12] Patane, L., Sciutti, A., Berret, B., Squeri, V., Masia, L., Sandini, G., & Nori, F. (2012, June). Modeling kinematic forward model adaptation by modular decomposition. In 2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob) (pp. 1252-1257). IEEE.
[ROJ23] Rojík, A. (2023). Real-time teleoperation of a robot arm for self-contact. MSc. thesis, Faculty of Electrical Engineering, Czech Technical University in Prague.

Name and workplace of master’s thesis supervisor:

doc. Mgr. Matěj Hoffmann, Ph.D. Vision for Robotics and Autonomous Systems FEE

Name and workplace of second master’s thesis supervisor or consultant:

Sergiu Tcaci Popescu, Ph.D. Vision for Robotics and Autonomous Systems FEE

Date of master’s thesis assignment: **25.01.2024** Deadline for master's thesis submission: __________

Assignment valid until: **21.09.2025**

doc. Mgr. Matěj Hoffmann, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I am grateful to Matěj Hoffmann and admire his work in organizing and running the “Cognitive and Humanoid Robotics laboratory”, without whom this work would not have been possible.

Thanks also to Sergiu Tcaci Popescu and Jason Khoury for their incredible support and guidance throughout my thesis.

Finally, I would like to express my heartfelt thanks to my entire family, who have made it possible for me to achieve all that I have now.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských prací.

V Praze dne 24. května 2024

. . . .
Oleg Baryshnikov

Abstract

This thesis focuses on developing an application for studies based on articles [1] and [2], which investigate the role of tactile sensations, proprioceptive sensations, and signals to perform voluntary movements in constructing the spatial perception of human beings.

To decouple all of these signals and study how each contributes to constructing spatial perception, a replication of the experiment from [1] and [2] was conducted. During experiment trials, the participant's left-hand movements were tracked by the motion capture system, and the Kinova Gen3 robotic arm was real-time controlled by the tracked participant's hand, providing 1D tactile extent to the participant's right arm.

The experiment replication required the application to operate the robotic arm and motion capture system, synchronize them, gather data from them and participants, control experiment blocks and trials, and provide relevant information via UI. Development of such an application was started by Adam Rojík in his diploma thesis [3], but not all required features for a complete experiment replication were implemented.

In the context of the current work, I extended the functionality of the Adam Rojík application, improved the code structure and software architecture, integrated unit tests, provided documentation to simplify future use and development processes of the application, and assisted the supervisors in replicating the experiments from articles [1] and [2].

Keywords: real-time teleoperation, self-contact, human-robot interaction, spatial perception, c++, software architecture, motion capture

Supervisor: Doc. Mgr. Matěj Hoffmann, Ph.D.

Abstrakt

Tato práce se zaměřuje na vývoj aplikace pro studie založené na článcích [1] a [2], které zkoumají roli hmatových vjemů, proprioceptivních vjemů a signálů k provádění dobrovolných pohybů při vytváření prostorového vnímání člověka.

Aby bylo možné všechny tyto signály oddělit a prozkoumat, jak každý z nich přispívá ke konstrukci prostorového vnímání, byla provedena replikace experimentu z [1] a [2]. Během pokusů experimentu byly pohyby levé ruky účastníka sledovány systémem snímání pohybu a robotická ruka Kinova Gen3 byla v reálném čase řízena sledovanou rukou účastníka, což poskytovalo 1D hmatové signály pravé ruky účastníka.

Replikace experimentu vyžadovala, aby aplikace ovládala robotickou ruku a systém snímání pohybu, synchronizovala je, shromažďovala od nich a od účastníků data, řídila bloky experimentu a pokusy a poskytovala příslušné informace prostřednictvím uživatelského rozhraní. Vývoj takové aplikace zahájil Adam Rojík ve své diplomové práci [3], ale ne všechny požadované funkce pro kompletní replikaci experimentu byly implementovány.

V rámci této práce jsem rozšířil funkčnost aplikace Adama Rojíka, vylepšil jsem strukturu kódu a architekturu softwaru, integroval jsem unit-testy, zajistil jsem dokumentaci pro zjednodušení budoucího používání a vývoje aplikace a pomohl jsem vedoucím při replikaci experimentů z článků [1] a [2].

Klíčová slova: teleoperace v reálném čase, sebe-kontakt, interakce člověk-robot, prostorové vnímání, c++, softwarová architektura, snímání pohybu

Překlad názvu: Teleoperace robotické paže v reálném čase pro manipulaci sebelokalizace u lidských účastníků

Contents

Chapter 1

Introduction

1.1 Motivation and main goals

Spatial awareness is one of the most important perceptions that significantly contributes to survival of many living organisms on our planet. Thoughts of how different species and human beings in particular perceive space around them occupied the minds of thinkers and scientists a long time ago and eventually led to the formation of several views and theories concerning this theme.

There are several signals that contribute to spatial awareness perception; this diploma thesis will focus only on three of them: tactile sensation, proprioceptive sensations, and signals to perform voluntary movements. Nowadays, there are theories that can explain how the signals mentioned above are formed. For example, generally accepted theories for tactile sensation postulate that it is formed in the somatosensory cortex, which receives action potentials from tactile receptors providing sensations of touch, pressure, and vibration. Even when we know how each signal works separately, it still remains unclear how those signals work together to form spatial perception, which we rely on in everyday life.

This could be studied by investigating the self-touch process, which involves all three signals: tactile, proprioceptive, and voluntary movement. The latest research on how those three signals influence spatial perception during the self-touch process was performed by Antonio Cataldo, Lucile Dupin, Harriet Dempsey-Jones, Hiroaki Gomi, and Patrick Haggard, and described in their two articles: “Sensorimotor signals underlying space perception: An investigation based on self-touch” [1] and “Interplay of tactile and motor information in constructing spatial self-perception” [2].

Those two articles contain descriptions of two experiments with identical setups but slightly different parameters. In these experiments, a participant interacts with two robots: he holds the first robot with their right hand and moves it freely from a certain start point to a certain end point. At the same time, the second robot repeats the participant’s right hand movements and touches the participant’s left hand, and the length of touch is not always equal to the participant’s left hand movements and may vary. At the end of the trial, participants must evaluate the extent of movement or touch. Details of these experiments are provided in Section 2.1.

These experiments may be subject to the following criticism: during the common self-touch process, such as when a person styles their hair, he freely decide when to end their movement instead of being interrupted during its production. However, in the experiments described in the articles, a participant's active movement is restricted by virtual walls, and the end position is strictly set. Consequently, a participant cannot decide where he might end movement as he would during normal voluntary movement. This leads to the difficulty of making conclusions about the influence of voluntary movement commands on our spatial perception during self-touch, and the role of voluntary motor commands remains unclear.

Avoiding that voluntary movement restriction is the main motivation of the experiment's replication that will be covered at this diploma thesis and especially in Chapter 3.

To repeat experiments, the setup with one Kinova Gen3 robot and the Qualisys motion capture system is used. Using the motion capture system instead of a second robot to track participant movements allows participants to move freely, as during common voluntary movements, in any dimension. Their position will be tracked by markers attached to the arm, and nothing will restrict participants in an ideal scenario. However, in the experiments described in articles [1] and [2], participants made their movements only in 1D, and to compare the results of the experiment replication with those of the previous ones, it is required to restrict participant movement in a way that allows them to move only in 1D during the experiment replication.

This setup requires an application that will operate the robot and motion capture system, synchronize them, gather data from them and participants, control experiment blocks and trials, and provide relevant information about them via UI. Adam Rojík started developing such a tele-touch application while writing his diploma thesis [3]. His application provides an interface for real-time teleoperation of the Kinova Gen3 robot via feedback from the Qualisys motion capture system in 1D and a simple graphical UI that allows experimenters to run experiment blocks.

However, a graphical UI to configure the experiment, a complete familiarization block for participants, logging of the data for evaluating the experiments are required for the experiment replication but were not implemented. Furthermore, the application that allows us to control robots using information from the motion capture systems can be used for future experiments; therefore, documentation, clean architecture, unit tests, and logic to control the robot in at least 2D extension are also needed.

The main goals of this work are to extend the functionality of the legacy tele-touch application, improve the code structure and software architecture, integrate unit tests, provide documentation to simplify future use and development processes of the application, and to replicate the experiments from articles [1] and [2].

1.2 Structure of the thesis

This thesis consists of the following chapters:

1. Introduction – contains the motivation, a quick overview of psychological aspects related to this work, related works, the legacy application, and the main goals of the thesis.
2. Context Analysis – contains a description of previous experiments [1] [2]: methods, setup, and results; as well as methods and setup for the experiment replication.
3. Experiment Replication – contains a description of the experiment replication. For the reader's convenience, it comes immediately after the analysis of previous experiments, the description of the robot and the motion capture system, preceding the design, implementation, and risk assessment chapters.
4. Legacy Tele-touch Application Analysis – contains an analysis of the legacy application's content related to the assignment of this diploma thesis, reasons for improvement or reworking it, as well as the legacy application's disadvantages that were found and must be fixed in parallel with implementing logic from the thesis assignment.
5. Software Application Design – contains specifications in a form of informal textual description of the logic that must be implemented during the thesis, together with a formal description provided as a set of functional requirements.
6. Software Application Implementation – contains a description of how the logic specified in the "Software Application Design" chapter was implemented and a showcase of it. Also it contains a description of the unit-tests that were designed and implemented in the application and a project documentation that was provided for the newly implemented parts of the application.
7. Risks assessment – contains an analysis and evaluation of possible risks that may occur during the experiment.
8. Conclusion – contains a high-level summary of what was accomplished.

Chapter 2

Context analysis

2.1 Experiments for investigation of interplay of tactile and motor information in constructing spatial perception

There are three theories that might propose controversial approaches to prediction of how motor (signals to perform voluntary movements), movement and sensory information influence our perception during self-touch. First one is Lotze's theory of local signs that postulates that motor information should have a logical priority over tactile information during self-touch [4]. Optimal integration theories assume that signals are independent and are weighted according to reliability at an integration stage when spatial perception is forming [5]. Another group of theories purpose completely independent spatial representations for movement and for tactile sensation [6] [7]. [1] [2]

Correctness of those three groups of theories in regard to self-touch process were covered in two articles: "Sensorimotor signals underlying space perception: An investigation based on self-touch" [1] and "Interplay of tactile and motor information in constructing spatial self-perception" [2] that investigate how motor, proprioceptive, and sensory information influence spatial perception during self-touch. Authors of articles performed several experiments with two linked robots where one robot was responsible for tactile part and other one for motor part to decouple motors from sensory components and investigate movement interference with tactile extent perception, and vice versa [1] [2]. The experiment setup from the [1] article is shown at the Figure 2.1.

Active motion and passive motion conditions were used to understand the role of the voluntary movement commands. A participant holds the first robot by his right-hand and moves it by himself in active condition, in passive condition participant hand movements were performed by experimenters. At the same time, the second robot touches the participant's left hand, and the length of the touch differs between the experiments described in the articles [1] and [2]. It equals to a fixed value from a set of 6, 8, or 10 cm in experiments from [1], or it corresponds to the length of the participant's movement and the gain applied to it in experiments from [2]. At the end of the experiment trial, participant must evaluate either the extent of movement or the extent of touch. The combination of active/passive conditions and movement/touch extent forms four experiment blocks: AM, AT, PM, PT.

Let's assume that movement and tactile information have completely independent roles in the process of forming spatial perception and consider the passive movement block ("D") in the experiment results from the article [1] shown in Figure 2.2. In such a case, if

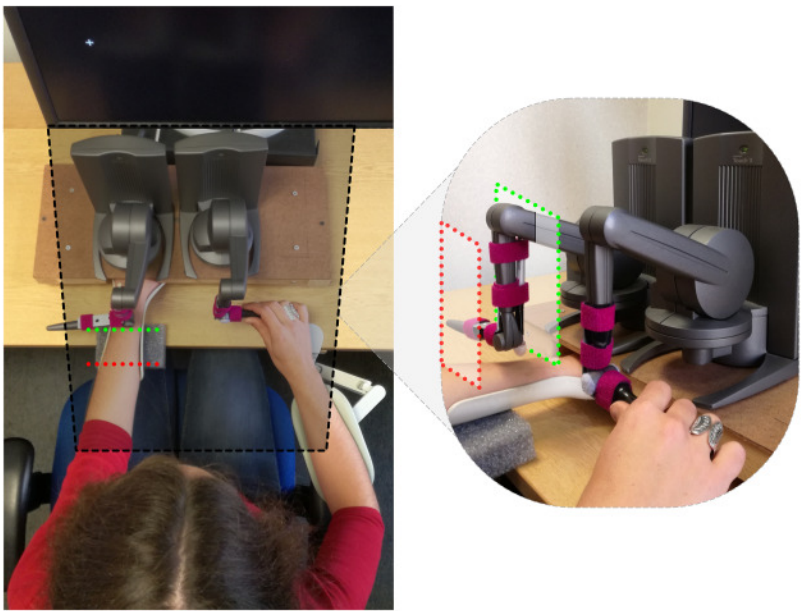


Figure 2.1: The self-touch experiment setup. [1]

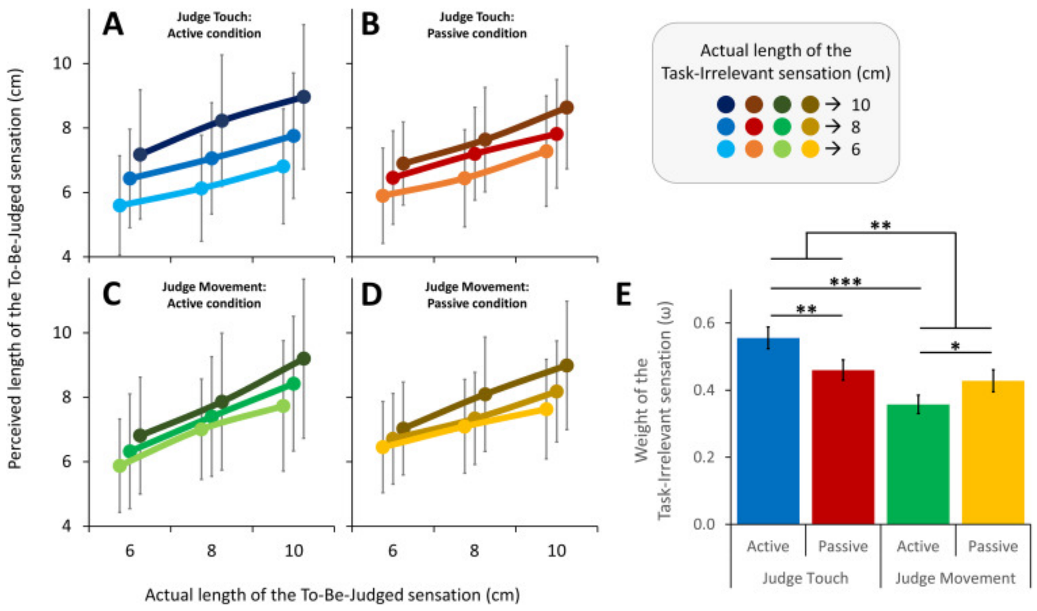


Figure 2.2: Results from the article "Sensorimotor signals underlying space perception: An investigation based on self-touch". [1]

experimenters performed participant's hand movement with any length from predefined set 6, 8, or 10 cm, then whatever touch the participant felt, touch extent must not affect the movement judgments; therefore, all lines that corresponds to different lengths of touch extent in the section "D" must overlap each other and be essentially the same. The same is for the passive touch block and section "B" in the results. However, in sections "B" and "D", we see that the lines are not the same. Let's also assume that the signals for the active movements don't affect and interfere with either the movement or the tactile signals in constructing spatial perception; then, in active conditions when the signals for the active movements are present, participants must judge movement or tactile extent fully equally to the passive condition. However, we see differences between "A" and "B" blocks and "C" and "D" blocks, which represent active and passive conditions respectively.

Specific differences between lines that denote different lengths of movement/touch extent in the "A", "B", "C" and "D" blocks show that movement extent strongly interferes with tactile extent perception and vice versa, and motor signals dominate the construction of spatial percepts, but this dominance is not total [1]. Also, block "E" shows the analysis of data from the other blocks described above. This analysis show differences in irrelevant information impact on a to-be-judged signal between active and passive conditions and from this can be concluded that signals for producing voluntary movement are also involved in spatial perception forming process.

Experiment result data in article [1] and article [2] differ. Results from article [1] show less task-irrelevant information impact during active movement of "judge movement" task while results from article [2] show the opposite. That may be caused by the difference of how the length of touch extent is formed in experiments. In the experiment from article [1], the results of which we considered above, the length of touch extent always equals to one of the value from the set of 6, 8, or 10 cm. In contrast, length of touch extent in experiments from article [2] equals to the length of the movement and the gain from set 0.(6), 1 and 1.5 applied to it; in other words, it is equal to the length of the movement multiplied by one of values from set of 0.(6), 1, 1.5.

Additionally, during all experiments participants' active movement is restricted by virtual walls and participants cannot decide where they want to stop their movement as during the normal voluntary movement. This leads to the difficulty of making conclusions about the influence of the voluntary motor commands on our spatial perception during the self-touch and the role of the voluntary motor commands remains unclear.

According to two facts described in the previous paragraph the experiment might be replicated to clarify task-irrelevant information impact during active movement of "judge movement" task with improvement of participants' ability to choose where he want to stop their voluntary movement to investigate the role of voluntary motor commands.

To replicate experiments setup with one robot and the motion capture system was used. The motion capture system doesn't require any virtual walls for measuring participants movements that introduces the ability for the participant to choose when he want to stop his movement, therefore the purely voluntary movements can be studied. To clarify task-irrelevant information impact during active movement condition, length of touch extent during the experiment replication will be equal to the length of the movement multiplied by one of values from set of 0.(6), 1, 1.5.

This setup requires the application that will operate the robot and motion capture system, synchronize them, gather data from them and participants, control blocks (AT, AM, PT, PM) and trials and provide relevant information about them by GUI. The development of such an application was started by Adam Rojík [3] in his thesis “Real-time teleoperation of a robot arm for self-contact – master thesis”. However, not all the required logic was implemented for a complete replication of the experiment, and several improvements and modifications need to be made, as described in Chapter 4. Adam Rojík’s application will be called the “legacy application” or “legacy tele-touch application” in the context of the current work. The description of the results of the experiment replication with the updated application is presented in Chapter 3, following this chapter for the reader’s convenience.

2.2 Kinova Gen 3 robotic platform

The Kinova Gen3 robot with 7 degrees of freedom is one of the main components of the experimental setup. The robot consists of a base, seven actuators, and the gripper as it shown in Figure 2.3.

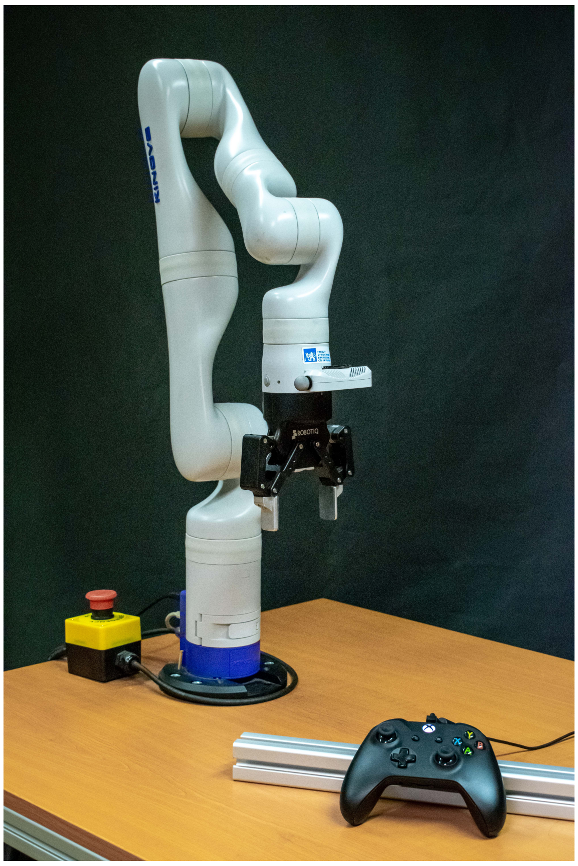


Figure 2.3: The Kinova Gen3 robot.

The robot can be controlled directly by an Xbox gamepad, which always takes priority over other control modes, or by the API with multiple servoing modes. A servoing mode is a modality through which commands are transmitted to robot devices during operation [8].

There are three servoing modes in total:

1. High-level servoing: the default mode where the user communicates with the robot's base. In this mode, the robot control library is involved to perform inverse kinematics computations for cartesian inputs and ensure safety by controlling collisions, boundary box, and other parameters.
2. Low-level servoing: in this mode, the user communicates with the robot's base, which provides faster API methods because the robot control library is not involved. Therefore, the user controls the robot by sending certain joint positions of the robot's actuators and must ensure safety and inverse kinematics by themselves.

3. Low-level bypass servoing: this mode allows the user to directly communicate with the robot's actuators by sending the position of joint angles to them. The robot control library is not involved.

In order to ensure minimal and imperceptible delay when a participant's hand is being tracked and the robot repeats the participant's movements, the legacy application uses the low-level servoing mode to communicate with the robot. The legacy application works with Cartesian coordinates to calculate the required position of the robot, checks if boundary and speed limits are not violated, and then performs inverse kinematics computations using the QuIK library to obtain joint angles for the robot's actuators. After that, it checks for any forbidden angles and finally sends the joint angles to the robot's base. These methods for ensuring safety will not be changed during the application changes in the context of this diploma thesis.

2.3 Qualisys motion tracking system

The main Qualisys motion tracking system setup consists of: at least two motion capture cameras, the calibration kit, a set of reflective markers, the computer or laptop with Qualisys Track Manager software installed on it, and power and data cables that connect the cameras and computer or laptop. The minimal Qualisys motion tracking system setup is represented in Figures 2.4 and 2.5.

Before using the motion capture system, the Qualisys Track Manager needs to determine camera positions relative to each other in order to correctly track markers. This must be done by performing a calibration process with the calibration kit. The calibration kit consists of the calibration triangle and the calibration wand, with reflective markers fixed in a certain way so that the distances between them are known. During calibration, the calibration triangle must lay on a surface inside the zone that will be tracked, while the calibration wand is actively moves around the tracking zone. After a successful calibration, the motion capture system can be used to track markers in the tracking zone.

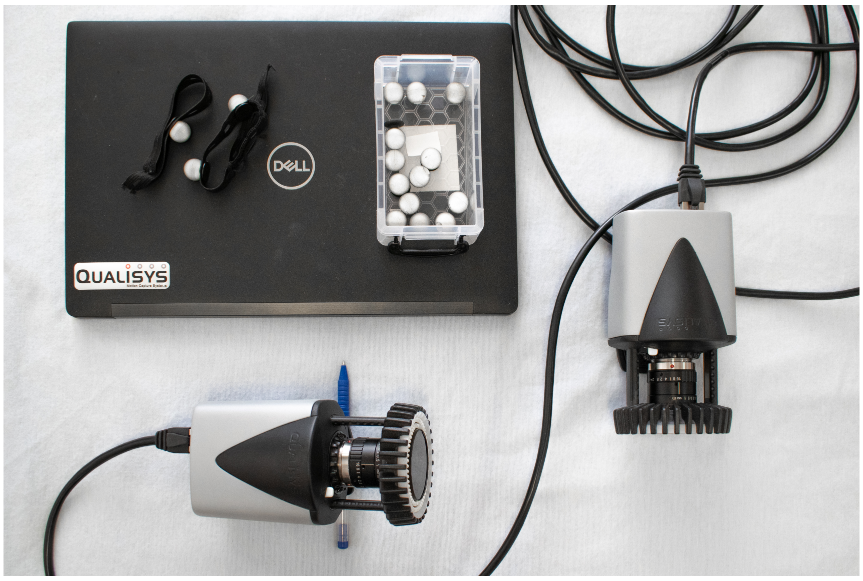


Figure 2.4: Two miqus motion capture cameras, power and data cable, markers and laptop with the Qualysis software.

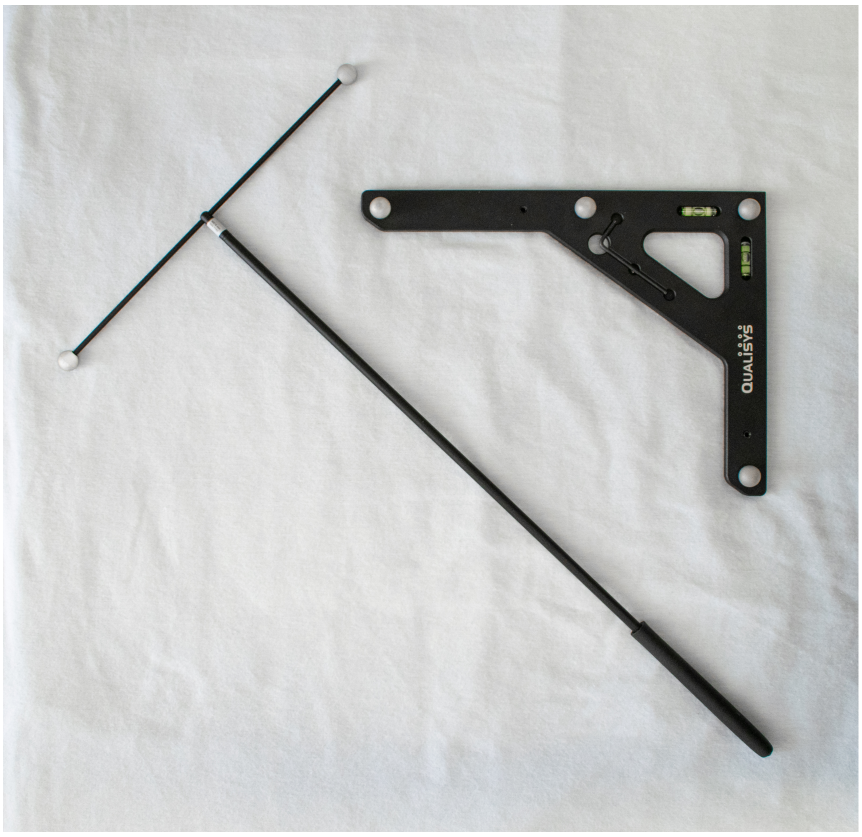
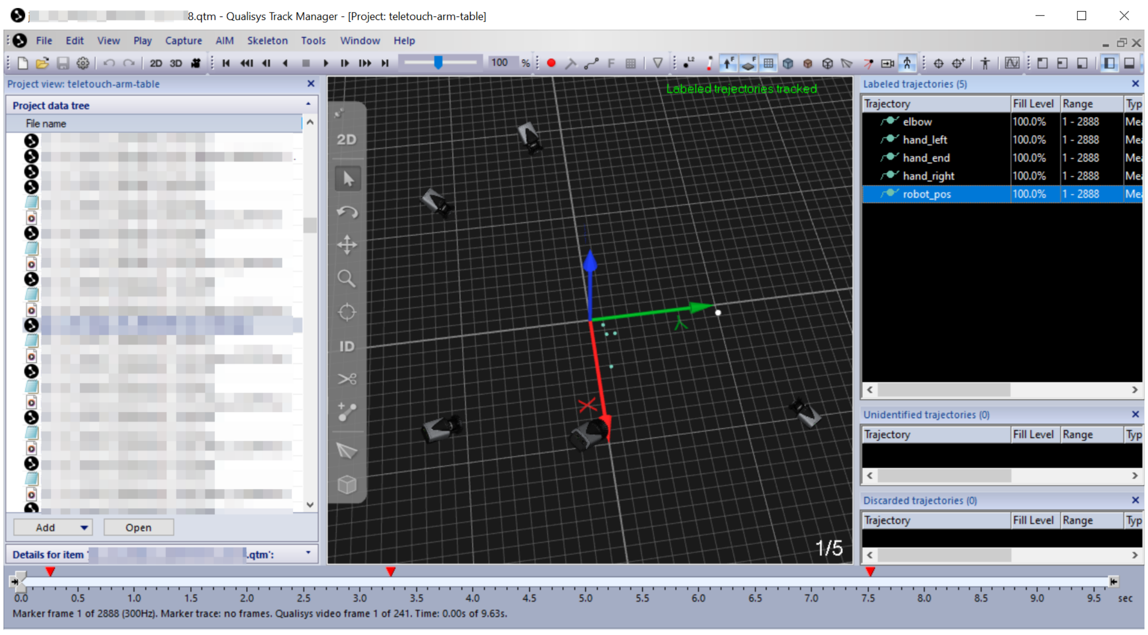


Figure 2.5: The motion capture system calibration tools.

The following terms may be used when working with the motion capture system:

- Measurement: A separate file that must be created in the motion capture system project in order to start a capture and work with the information that was received during the capture.
- Capture: The period when a measurement is open, and the motion capture system tracks and saves the positions of markers in the tracking zone.

If the motion capture system is used to track markers with the same structure, such as markers always placed on certain parts of the hand, it may be useful to create an Automatic Identification of Markers (AIM) model. This model can then be applied to any measurement that captures similar motions with the same marker set [9]. Markers in the AIM model can be named, and their positions can be retrieved using the Qualisys API [10] during a capture. The set of tracked markers with the applied AIM model in an opened measurement is shown in Figure 2.6.



[redacted]8.qtm - Qualisys Track Manager - [Project: teletouch-arm-table]

Project view: teletouch-arm-table

Details for item '[redacted].qtm':

Labeled trajectories (5)

Trajectory	Fill Level	Range	Typ
elbow	100.0%	1 - 2888	Me
hand_left	100.0%	1 - 2888	Me
hand_end	100.0%	1 - 2888	Me
hand_right	100.0%	1 - 2888	Me
robot_pos	100.0%	1 - 2888	Me

Unidentified trajectories (0)

Trajectory	Fill Level	Range	Typ

Discarded trajectories (0)

Trajectory	Fill Level	Range	Typ

Marker frame 1 of 2888 (300Hz). Marker trace: no frames. Qualisys video frame 1 of 241. Time: 0.00s of 9.63s.

Figure 2.6: The Qualisys Track Manager view.

The positions of the markers during a capture with the AIM model applied on it are actively used by the legacy tele-touch application to track participants' hand movements and repeat them by the robot's movements.

Chapter 3

Experiment replication

While replicating the experiment, four Miqus motion capture cameras attached to tripods to track markers at a frequency of 300 Hz were used. The cameras were connected to a laptop with the Qualisys track manager application open, and this laptop was connected to the laboratory network. Another laptop connected to the laboratory network ran the tele-touch application, and an external DELL U2415 screen with an active screen height of 32.1 cm (1200px) was used to display the application GUI. The Kinova Gen 3 robot was turned on and connected to the laboratory network. The robot gripper was equipped with a brush with a long bristle. Additionally, there was a right-hand support allowing dynamic changes in horizontal arm position, a left-hand fixator to restrict participant movements in one direction, the wireless mouse that allows a participant to switch application scenes, four markers with elastic bands attached to the participant's hand, and one marker on the robot. Finally, a motion capture calibration kit was used for motion system calibration.

Before starting each experiment with a participant, the robot was rebooted, and the motion capture system was calibrated to ensure maximum data precision and experiment safety. An AIM model was generated, and since the robot's operating height was known and fixed in the settings file, the vertical brush position was adjusted to ensure participant comfort. The participant then placed markers on his left hand: one marker is placed on the elbow, two markers on the wrist, and one on the base of the little finger, took his seat, and attached his left hand to the hand fixator on the table. His right hand was placed on the table with the right-hand support, which was dynamically aligned horizontally so that the brush could equally touch the entire surface of the hand when the robot was moving. The participant's hand position during the experiment is shown on Figures 3.1 and 3.2.

While the participant performed the operations described below, the order of the four experiment blocks was defined, and participant info files were created by experimenters using the application GUI. The participant was then introduced to the instructions for the familiarization block, and the UI familiarization scene with familiarization block GUI components was displayed on the screen. The participant attempted to perform ten good familiarization trials to familiarize themselves with the experiment setup, learn to move his hand at a nearly constant speed, and utilize all the provided space for movement during experimental trials. The participant that is trying to make a good trial in the familiarization block is shown on Figure 3.3.

When the participant performed ten good familiarization trials, he was considered prepared, and the first experiment block from the predefined order began. Before each

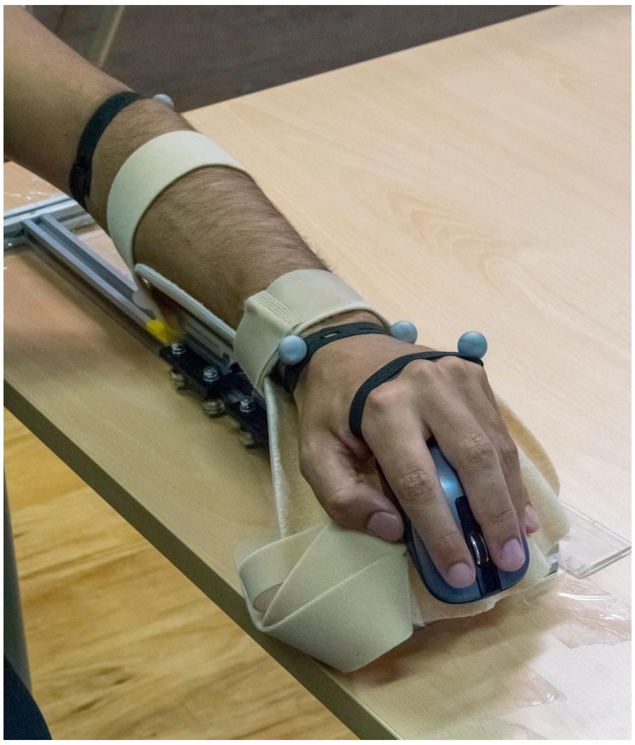


Figure 3.1: Participant's left hand during experiment.

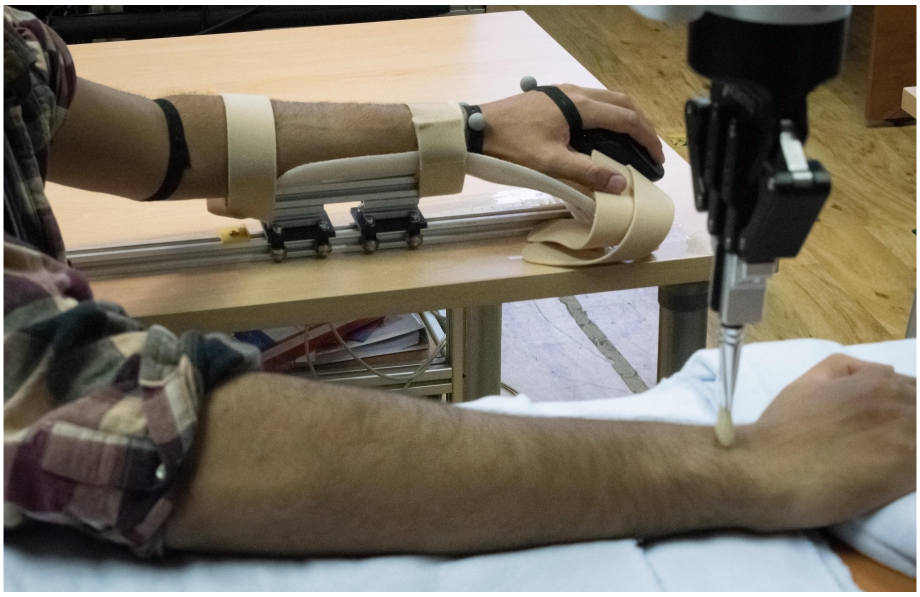


Figure 3.2: Participant's move his left hand and the robot repeat his movements.

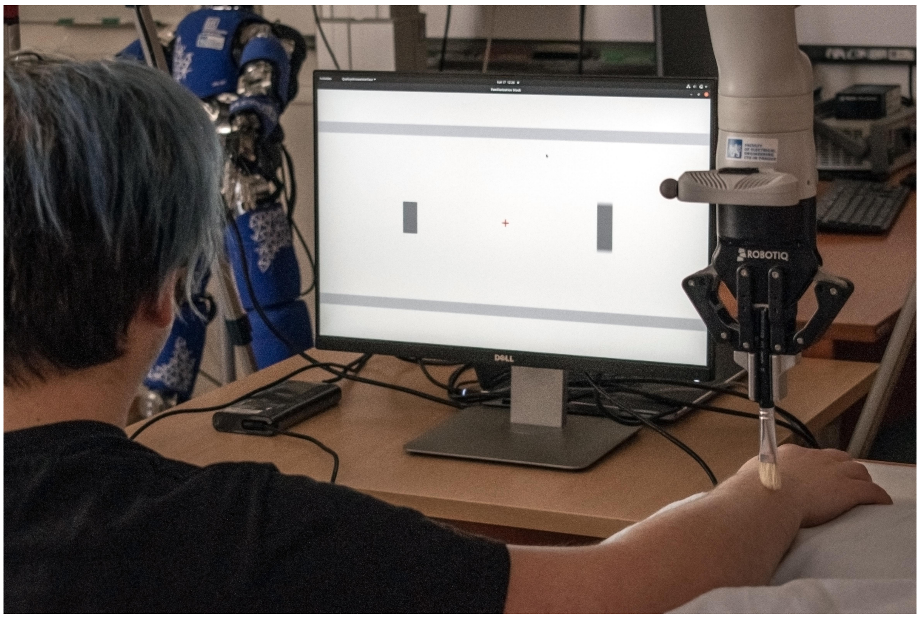


Figure 3.3: Participant is performing the familiarization trial.

experiment block, the participant received instructions and was asked to focus and evaluate the extent of touch or movement. The participant was unaware that the robot did not always replicate his movements 1:1 and that gain was applied. Before each block, customizable block settings were adjusted to match the participant arm length. Each experiment block consisted of 25 to 100 trials. After completing each block, the participant could choose to take a break or proceed to the next block in order, and the application was restarted. Once the last block in the order was finished, the experiment concluded.

Five participants took part in the experiment, interaction with one participant lasting from 1 hour to 2.5 hours. Approximately 750 experimental trials and 100 familiarization trials were performed. For each experiment, the following output files were obtained:

- Output files from the Qualisys motion capture system containing information about the positions of all five markers for each trial.
- Output files containing information about each experiment block performed.
- Output files with detailed information for each trial performed during the familiarization block.
- Output files with information about each trial performed during every experiment block.

Examples of each of these files are located in the ./cpp/output_data_example directory on the project's GitLab [11] and detail describtion of those files are provided in the Section 5.4.

The collected data is sufficient to begin future analysis, although data analysis is going outside of the scope of the current work.

Chapter 4

Legacy tele-touch application analysis

4.1 Architecture and design

During the analysis of the legacy application, several design, style, and architecture disadvantages described in the text below were identified, and after analyzing these disadvantages, several recommendations were made to eliminate them.

The legacy application contains 7 non-external-libraries files with the source code. All of them are shown in Figure 4.1, which includes a non-standard diagram that shows the legacy application files, relations between them, and information about the structures and classes they contain. The "includes" relation between some files A and B on this diagram means that file A contains the "`#include B`" directive.

The Working Draft, Standard for Programming Language C++ [12] which can be found on GitHub [13] says: "The text of the program is kept in units called source files in this document. A source file together with all the headers and source files included via the preprocessing directive `#include`, less any source lines skipped by any of the conditional inclusion preprocessing directives, is called a translation unit.". According to the standard draft, these translation units compile completely independently from each other. The legacy application has only one source file – `main.cpp`, and will have only one translation unit formed from the `main.cpp` file and all files that it directly or indirectly includes. If changes occur in any of those files, the entire translation unit must be recompiled, leading to longer compilation times when any part of the application source code changes. Using a group of logically separated source files, such as `camera_reader.cpp`, `robot_controller.cpp`, etc., will significantly reduce compilation time after source code changes.

The all source code of the legacy application is located inside `*.h` files. A `*.h` file, commonly known as a header file, typically contains declarations of types and functions, while `.cpp` files commonly contain their definitions. Bjarne Stroustrup, the inventor and developer of C++ language, addresses this separation in his book "A Tour of C++" [14], stating: "This can be used to organize a program into a set of semi–independent code fragments. Such separation can be used to minimize compilation times and to strictly enforce separation of logically distinct parts of a program (thus minimizing the chance of errors)". Therefore, to reduce the application compilation and recompilation times and logically distinct declaration and definitions to minimize the chance of errors, the application source code contained within `.h` files must be separated. Definitions of types and functions should be placed in `.cpp` files, while declarations should remain in `.h` files.

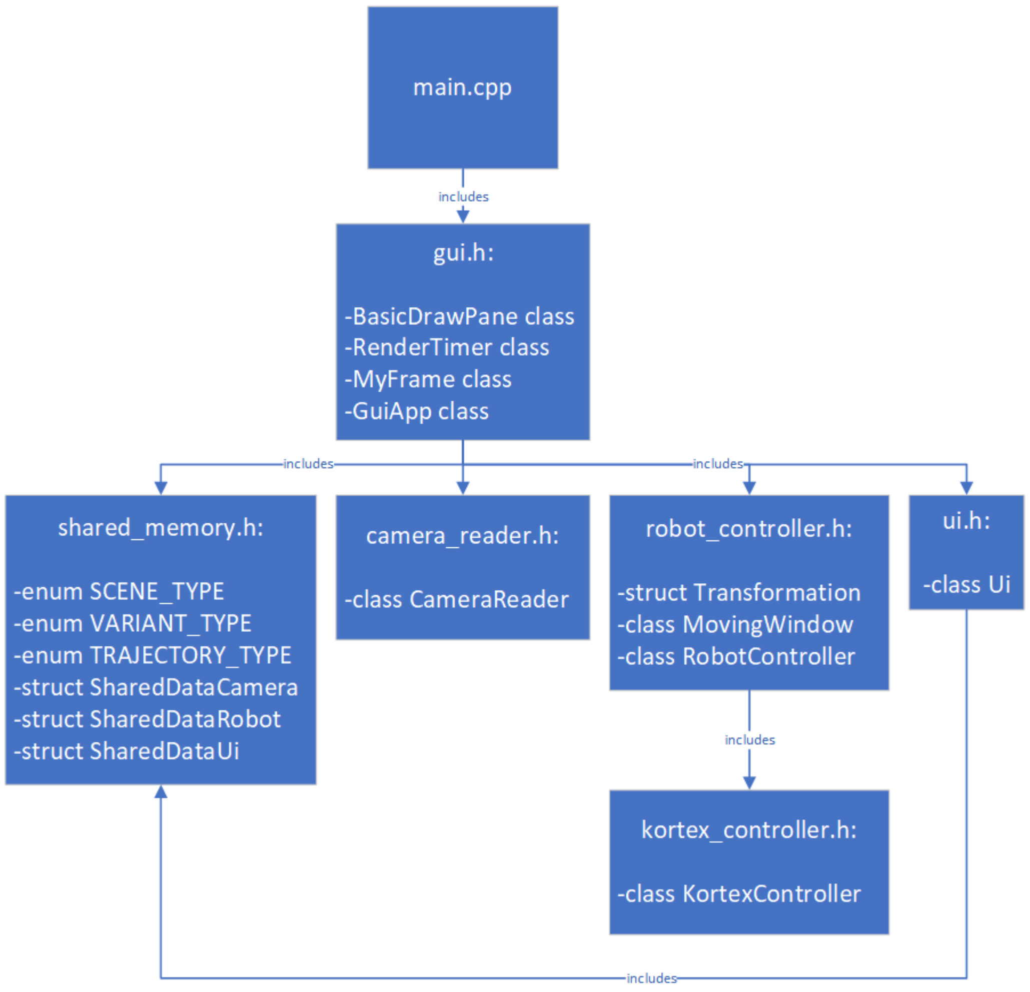


Figure 4.1: Legacy files relations diagram.

The application's classes are grouped by files. While `camera_reader.h`, `robot_controller.h`, `ui.h` and `kortex_controller.h` files each contain only one class that is semantically separated from others, the `gui.h` file contains several classes with various purposes that could theoretically be reused in classes that are located in other files. However, grouping classes by files leads to less reusability and more complex navigation compared to grouping them by folders. For instance, the `RenderTimer` class may be used in GUI classes that are located in other files, but including the entire `gui.h` file, which also contains less-reusable classes like `MyFrame`, is redundant. Additionally, it may be challenging to locate the `RenderTimer` class because there is no indication that `gui.h` contains it. To avoid redundancy and improve navigation in the project, it is better to have one class per file and semantically group those files into separate directories.

All the UI state processing logic is mainly located in the "`update`" method in the "`Ui`" class, while the UI component drawing logic is located in the "`render`" method of the "`BasicDrawPane`" class. Separating data processing logic and drawing logic is a useful technique because it contributes to writing more isolated unit-test methods and enables switching graphical libraries by simply changing one GUI class to another. However, these classes are designed in such a way that the "`update`" method (298 lines of code) and the "`render`" method (131 lines of code) are large and include logic for processing every single GUI application state and drawing every single GUI element. When encountering an issue with how a rectangle is drawn during the experiment on the screen, it is much easier to locate and fix the error in the small "`DrawRectangle`" method than within the large general "`render`" method. As the application grows, these methods will grow too, making it more difficult to implement new features and locate errors in the GUI. Therefore, the "`render`" and "`update`" methods must be separated into several smaller, semantically separated methods.

The GUI logic is spread between the `ui.h` and `gui.h` files, the purposes of which are not clear from their names. The names of the `BasicDrawPane` and `MyFrame` classes are quite incorrect. The `BasicDrawPane` contains the logic for drawing all scenes in the application, so calling it "Basic" does not accurately reflect its complexity, and the name "`MyFrame`" means nothing without context. As the application grows, such ambiguous class and file names will lead to confusion because programmers will struggle to quickly determine the purpose of each class. Class and file names should indicate the semantic purpose of the class.

The GUI, camera, and robot classes are designed to communicate with each other using shared data structures and mutex locks. For example, if the GUI object wishes to indicate that the scene has changed, it modifies the "`scene`" variable in the "`sharedDataUi`" data structure. However, it remains unclear which object will read this variable and what consequences it will lead to. The lack of understanding of the effects of variable changes during editing of shared data structures leads to difficulties in future application support, development of new features, and debugging in case of errors. To avoid this, every communication between UI and non-UI components should be done by calling methods with names that represent their semantic.

Some parts of the application source code contain strong dependencies between unrelated classes. For example, when the `RobotController` class needs to determine which action it must perform, it reads the "`scene`" variable from the "`sharedDataUi`" shared data structure and performs different actions based on the information read. This imposes a requirement that the `RobotController` must know which scene the UI has and how to react to every particular UI scene, which prohibits the `RobotController` from being used in applications with other UIs. To change this, classes that are not responsible for UI logic must be independent from the UI classes.

To reduce compilation time, achieve better navigation within the project, ensure class reusability, minimize the chance of errors, and reduce the time required to implement new features or fix errors, the application architecture must be entirely reworked, and the application source code must be rewritten, taking into account the design and style recommendations described above.

4.2 Familiarization block

Every experimental block requires participants to be familiarized with the experimental setup and to be able to move their arm at approximately the same speed during every trial to reduce the intra-subject and inter-subject movement variability in order to get coherent data. Additionally, participants need to be taught that they may freely use all the range of possible movement extents provided for them that also means end their movement at any point on their right hand to ensure that their movements will be truly voluntary during the experiment and to obtain data for most available distance values. To satisfy these criteria, a familiarization block is used. In summary, the familiarization block has the following goals:

- Familiarize participants with the experimental setup.
- Teach participants to move their hand at a close to constant speed during experimental trials.
- Teach participants to use all the space provided for movement.

The logic of the familiarization block in the legacy application is primitive. During the legacy familiarization block, participants can perform an unlimited number of trials. Each trial consists of two scenes: the introduction scene informs the participant that the familiarization block has been selected, and after the participant clicks, the countdown scene occurs. Right after the countdown, the robot starts following the participant' left hand and touches the participant' right hand.

These two scenes only familiarize the participant with the process of robot manipulation. However, participants will not be able to learn how to perform movements at a close to constant speed and explore all the available space for movement. Because of this, the legacy familiarization block must be reworked by adding groups of new features that will help participants acquire these important experimental skills. Specifications for the new familiarization block are described in the Section 5.2.

4.3 Experiment parameters configuration

This section contains an analysis of how the legacy application works with participant data and which parameters it allows to change in experiments and familiarization blocks.

Before the experiment starts, structured information about a participant must be gathered and saved in a well-defined format. That information must identify by the participant's code name and must contain the participant's age, gender, order of the AT, AM, PT, PM blocks that will be performed during an experiment with the participant, and the length of the participant's forearm. This information will be used during further analysis of the data gathered during the experiment with the participant.

To prevent format violations and maintain consistency in participant data across all experiments, it is advisable to implement a user interface with functions for creating, editing, and accessing participant data within the application. However, the legacy application doesn't contain any logic for working with participant data.

Additionally, experiment and familiarization blocks have their own set of parameters that may be adjusted before the block starts, at the experimenter's discretion. In the legacy application, experimenters can only modify a few basic application settings, such as screen height in centimeters for correct GUI displaying, connection parameters for the motion capture system and Kinova robot, and certain safety parameters like maximum joints velocity and initial robot positions, by manually editing the `settings.ini` configuration file. Any other customizations to experiment and familiarization blocks can only be made by modifying the source code. This process requires specialists familiar with the source code and time for recompilation of the application.

An example of source for possible customizations is an editable vertical line used to provide participants with the possibility to evaluate extents in centimeters. The length of that judgement line is randomly chosen from uniform distributions at the start of the judgment scene. Participants can change the length of that line, and the final judgment will be equal to the length of the vertical line on the screen. Sometimes, experimenters may want to adjust the graphical line width for visually impaired participants, set boundaries for uniform distributions of the judgment line to match participant arm length, or specify the step of change in the judgment line.

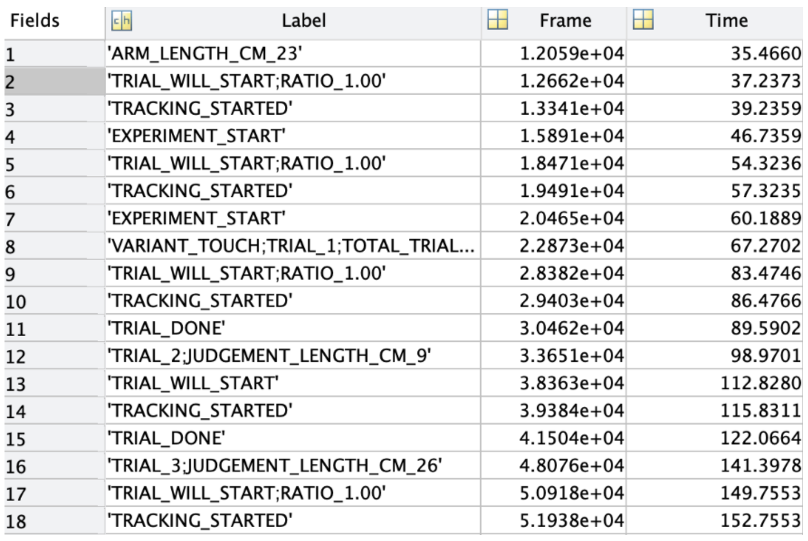
To reduce time of changing experiment and familiarization block parameters and to minimize the risk of errors that experimenters may encounter when working with participant data, a new application screen with functions for managing participant data and customizing experiment and familiarization block parameters must be developed. Specifications for that graphical UI to configure the experiments are described in the Section 5.3.

4.4 Data logging for future evaluation

The legacy application has only one output file that is created after the application closes. It is a file with information from the Qualisys motion capture system containing recording parameters and frames with coordinates that represent the position of every marker during the application work. Additionally, the legacy application sends a set of different events to Qualisys as a strings with information about scene or block switching, trial end, gain changing, arm length, and judgment line length. The output file also contains information about all received events connected to the frame when Qualisys received that event. An example of events that was logging during the legacy application run is shown on the Figure 4.2.

The legacy application's output data file includes information for all trials that occurred during the application work. During the data analysis process, the data must be separated by the trials to analyze each one separately, and this separation requires human resources and time.

The information from the motion capture system is not fully reliable because sometimes during the capturing process the motion capture system may lose some markers or detect



Fields	Label	Frame	Time
1	'ARM_LENGTH_CM_23'	1.2059e+04	35.4660
2	'TRIAL_WILL_START;RATIO_1.00'	1.2662e+04	37.2373
3	'TRACKING_STARTED'	1.3341e+04	39.2359
4	'EXPERIMENT_START'	1.5891e+04	46.7359
5	'TRIAL_WILL_START;RATIO_1.00'	1.8471e+04	54.3236
6	'TRACKING_STARTED'	1.9491e+04	57.3235
7	'EXPERIMENT_START'	2.0465e+04	60.1889
8	'VARIANT_TOUCH;TRIAL_1;TOTAL_TRIAL...	2.2873e+04	67.2702
9	'TRIAL_WILL_START;RATIO_1.00'	2.8382e+04	83.4746
10	'TRACKING_STARTED'	2.9403e+04	86.4766
11	'TRIAL_DONE'	3.0462e+04	89.5902
12	'TRIAL_2;JUDGEMENT_LENGTH_CM_9'	3.3651e+04	98.9701
13	'TRIAL_WILL_START'	3.8363e+04	112.8280
14	'TRACKING_STARTED'	3.9384e+04	115.8311
15	'TRIAL_DONE'	4.1504e+04	122.0664
16	'TRIAL_3;JUDGEMENT_LENGTH_CM_26'	4.8076e+04	141.3978
17	'TRIAL_WILL_START;RATIO_1.00'	5.0918e+04	149.7553
18	'TRACKING_STARTED'	5.1938e+04	152.7553

Figure 4.2: Events from the legacy application output files containing all the information. Figure from the “Real-Time Teleoperation of a Robot Arm for Self-Contact” work [3].

non-existent markers in reflections from the robot or rails. Therefore, additional and more reliable sources of information are required. For example, the robot’s position could serve as one of these sources.

Information about different legacy application values that the legacy application tries to save are represented as part of the names of events in the motion capture system capture that are commonly used to “mark something that is happening” [9] but not to store values. Storing complex data structures in a set of events will lead to a more complex parsing process during the output data analysis.

The graphical UI to configure the experiments logic, the need for which was described in the previous section, assumes that the application will manage participant data and customize experiment and familiarization block parameters, thus requiring saving this information in order to analyze it later.

To reduce data parsing time and complexity, there should be a separate motion capture system output file for each trial. Additionally, application information should be logged in files separate from the Qualisys motion capture output data files. These files must include participant data, customizable experiment and familiarization block parameters, as well as information of the robot’s position during trials.

Chapter 5

Software application design

5.1 The new application architecture design

To satisfy all recommendations provided in the architecture analysis Section 4.1, a new architecture has been created for the application. The description of the new architecture, along with its main advantages over the previous program structure, is provided below as description of several semantically separated application components with methods and data that they can have.

All application definitions of types and functions will be placed in `.cpp` files and their declarations will be placed in `.h` files. Each class representing an application component will be represented by its own `.cpp` and `.h` files with names that reflect its purpose, and these files will be grouped by directories with appropriate names to ensure comfortable navigation for a programmer.

To preserve the possibility of easier UI library changes and to separate data processing and drawing logic, the application's UI logic has been divided into three main components: view, view implementation, and presenter. Each semantically separated part of the application UI will consist of its own presenter, view, and view implementation(s). For example, the application will have `ExperimentView`, `FamiliarizationView`, and `SetupView`, along with their implementations and corresponding `ExperimentPresenter`, `FamiliarizationPresenter`, and `SetupPresenter` classes. This separation helps prevent the consolidation of all logic in the GUI and UI classes.

A view is an application component represented by an interface that defines the necessary UI-drawing functions used by the presenter. These functions must be implemented in the view implementation. Utilizing this interface allows the application to have several different implementations of the same view, for example with different graphics libraries, which will work with the same presenter.

A View Implementation is an application component represented by a class that implements the view interface. It can utilize graphical libraries and utilities to draw the UI as directed by the presenter. The View is designed to be maximally passive; it receives UI events and directly passes those events which can significantly affect the application state changes without processing them to the presenter by calling the presenter's methods and waits for presenter's commands. Additionally, the View can only store the data needed to draw the user interface and to utilize UI libraries. The logic behind the View can be distributed among individual graphical components at the programmer's discretion.

A presenter is an application component represented by a class that includes methods and data to handle and react to events that can significantly affect application state changes coming from the view or services. The presenter should not strictly depend on any graphical libraries or components; the view is responsible for work with graphical elements. The presenter can independently process received as a result of reacting to events from the view, or it can utilize services and utilities for this purpose. Additionally, the presenter calls functions of the view interface to draw the application state changes.

The presenter and view components are essentially the main components for drawing semantically separated application UI parts. Services and utilities are designed with the possibility of being used outside the application context, for example, in other applications, and do not include application UI logic. This ensures that classes not responsible for UI logic are independent from the UI classes.

A service is an application component represented by a class that includes semantically separated non-UI methods, with names that reflect its purpose. These methods can be called in various presenters and other application services. A service encapsulates the data required for the operation of these methods and can also modify data that can be used outside the application context, such as files on ROM, the motion capture system state, robot states, and others. Only services can create new threads, with the operation logic described in workers, and communicate with them using shared data. This ensures that every communication between UI and non-UI components is done by calling methods with names that represent their semantics.

A worker is an application component represented by a class that includes methods and data designed to create new threads and work within them. Workers can only appear as part of some services and may include utilities and services to perform certain operations.

A utility is an application component represented by a class that includes semantically separated methods, which can be used in various presenters, views, services, workers and other utilities. A utility can also encapsulate the data required for its methods, but can modify data that can be used outside the application context only as a part of a service and cannot spawn new threads.

Separating the application into small, semantically independent parts such as view implementations, views, presenters, services, workers, and utilities not only provides the benefits described above but also reduces the time required to implement new features and identify and fix errors. This is because it is easier to find errors or implement methods in small, semantically separated classes than in one large, general-purpose class.

The application components described above, along with all their connections, are graphically represented in Figure 5.1. In the non-standard diagram, application components are represented as rectangles. The "includes" relation between some components A and B in this diagram means that file A contains the "`#include B`" directive. The "may include" relation between some components A and B means that file A may contain the "`#include B`" directive. An arrow pointing to one rectangle of component A indicates that some component includes or may include only one such component A. An arrow pointing to one rectangle and one rectangle behind component A indicates that some component includes or may include many such components.

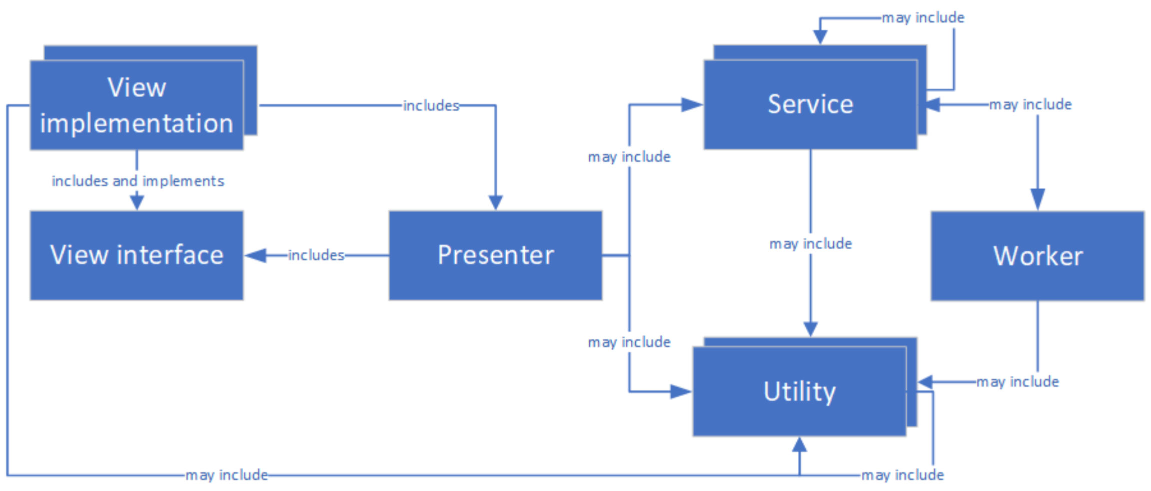


Figure 5.1: The new application architecture.

Specific and concrete examples of application components are given in the implementation Chapter 6.

5.2 Familiarization block - design

The description and specifications that was used for further implementation of the new familiarization block is provided below.

The Familiarization block was designed to fulfill the following goals:

- Familiarize participants with the experimental setup.
- Teach participants to move their hand at a close to constant speed during experimental trials.
- Teach participants to use all the space provided for movement.

In order to achieve those goals, a participant is asked to perform several “good trials” during the familiarization block. To perform a “good trial”, the participant must move their hand by a distance that is approximately equals to a randomly chosen length for the current trial and at a speed within the lower and upper bound speeds defined by experimenters. To assist the participant, four lines must be displayed on the screen: one dynamically representing the length of movement performed by the participant, a second serving as a reference line continuously increasing on the screen at speeds within the lower and upper bounds, until it becomes equal to randomly chosen length for the current trial, to help the participant in imagining the required speed to perform a “good trial”. The third and fourth lines represent the randomly chosen length as the gray zone that the participant must achieve to perform a “good trial”.

Before the new familiarization block starts, the experimenter chooses values for parameters of the entire familiarization block, such as the good trial speed lower bound, good trial speed upper bound, required number of good trials, boundaries for the uniform distribution of the final reference line length, and others parameters that are defined in the Section 5.3.

During the new Familiarization block, participants perform several trials with the robot and motion capture system involved. The setup is similar to the experimental setup: during trials, the participant moves their left arm, the robot repeats the participant’s movement, and touches the participant’s right arm – this familiarizes participants with the experimental setup.

At the start of every trial, the maximum length of the reference line for the given trial is chosen from a uniform distribution of final reference line lengths. To teach participants to use the entire range of possible movement lengths, during every trial participants must follow a continuously growing reference line and traverse a distance that approximately equals the maximum length of the reference line for the given trial. To teach participants to move their hand at a close to constant speed during experimental trials, participants need to move their hand within speed boundaries. If during a trial, the participant traverses a distance that approximately equals the maximum length of the reference line for the given trial and doesn’t cross the speed lower bound and speed upper bound, then the trial must be considered as a “good trial”. When participants perform a certain number of good trials, the familiarization block finishes. A formal and full description of the new familiarization block is provided below as a set of functional requirements.

The new Familiarization block consists of the following scenes: UI familiarization, Capture starting, Block information, 321+ countdown, Familiarization active trial, Familiarization trial evaluation, Camera data saving, and Familiarization completed. Each scene is a logically separated set of UI elements that the user sees on the screen. At any given moment, only one scene may be active. Possible scene transitions are illustrated in Figure 5.2. The scenes that need to be implemented are described as a set of functional requirements approved by experiment supervisors and provided in Table 5.1 below.

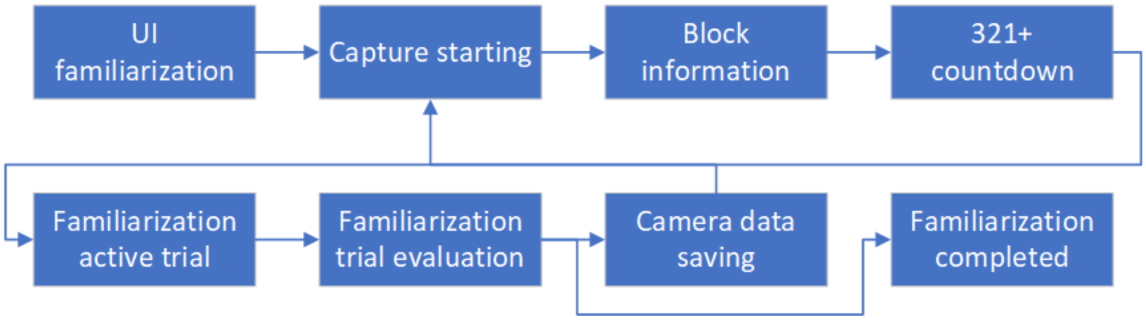

UI familiarization
Capture starting
Block information
321+ countdown
Familiarization active trial
Familiarization trial evaluation
Camera data saving
Familiarization completed

Figure 5.2: Familiarization block scene transitions.

Code	**Functional requirement**
FFR–1	The following parameters must be set by the UI before the “UI familiarization” scene starts: the participant speed lower bound, the participant speed upper bound, the reference line speed, the reference line acceleration and deceleration time, the smoothing rate, the transparency of lines and borders, the size of the gray zone, the number of good trials, the maximum reference line length, the minimum reference line length, and the width of the lines.
FFR–2	After the “UI familiarization” scene starts, the application must connect to the robot and send the command to move to the “participant’s right hand position”. This position is defined in the application configuration file. After the robot reaches that position, the application must send the command to start moving to a “position near the participant’s right hand”. At the “position near the participant’s right hand” robot must not touch the participant hand.
FFR–3	During the “UI familiarization” scene, a participant must see on the screen all UI elements from the “Familiarization active trial” scene and short description to them.
FFR–4	During the “UI familiarization” scene, a participant may click on the left or right mouse button to go to the “Capture starting” scene.
FFR–5	During the “Capture starting” scene, a participant must see on the screen the text “Capture starting”.
FFR–6	After the “Capture starting” scene starts, the application must connect to the motion capture system, open a new measurement, start a new capturing and wait until the robot reaches a “position near the participant’s right hand”.
FFR–7	During the “Capture starting” scene, if the application is connected to the motion capture system and a new capture is started and the robot reached the position, then the scene must switch to the “Block information”.
FFR–8	During the “Block information” scene, a participant must see on the screen the text that informs them that the familiarization block is running.
FFR–9	During the “Block information” scene, a participant may click on the left or right mouse button to go to the “321+ countdown” scene.

FFR–10	During the "321+ countdown" scene, the application must perform the "3", "2", "1" countdown and show it as the text on the screen.
FFR–11	During the "321+ countdown" scene, the application must send the command to the robot to move to the "participant's right hand position". It is assumed that at the "participant's right hand position" the robot touch the participant right hand.
FFR–12	During the "321+ countdown", when the countdown finishes, the scene must switch to the "Familiarization active trial".
FFR–13	At the start of the "Familiarization active trial" scene, the application must perform the following actions: it remembers the current participant's left hand position, which becomes the initial left hand position and then chooses the maximum length of the reference line for the given trial from a uniform distribution – after that trial begins.
FFR–14	During the "Familiarization active trial" scene, the robot must touch the participant's right hand and start to repeat the participant's left hand movement in 1D direction along the participant's hand.
FFR–15	During the "Familiarization active trial" scene, a participant must see two vertical lines on the screen: one of them is a reference line and another one is the participant line.
FFR–16	During the "Familiarization active trial" scene, the reference line must continuously grow up with the constant speed that equals to the reference line speed parameter until it reaches the reference line maximum length.
FFR–17	During the "Familiarization active trial" scene, the reference line has an acceleration period when the speed increases from zero to certain constant speed, a constant period when the speed is constant and a deceleration period when the speed decreases from constant to zero.
FFR–18	During the "Familiarization active trial" scene, the participant line must dynamically change to represent the current distance that the participant's left hand made from the initial left hand position.
FFR–19	During the "Familiarization active trial" scene, a participant must see on the screen two bold horizontal borders which form the trial gray zone that represent the reference line maximum length towards which the participant line and the reference line must tend.

FFR–20	During the “Familiarization active trial” scene, a participant must see a cross at the screen center.
FFR–21	During the “Familiarization active trial” scene, a participant may click the left or right mouse button to switch the scene to the “Familiarization trial evaluation”, thus ending the current trial. Participant’s and reference line speeds must be recorded during the trial with certain smoothing applied. If, during the movement at “Familiarization active trial” scene, the participant’s line speed didn‘t cross the certain speed upper bound and the certain speed lower bound, and the participant line reached the gray zone, then the current trial must be considered as an good trial.
FFR–22	At the end of the “Familiarization active trial” scene, the application must send commands to the robot to move away from the participant’s right hand and then to start moving to a position near the participant’s right hand.
FFR–23	During the “Familiarization trial evaluation” scene, a participant must see the following UI elements on the screen: the gray zone, the participant line, the feedback text, the speed graph and the text with information about total number of trials and the current good trial number. The feedback text must inform whether the trial was considered as an good trial and provide reasoning if not. The speed graph contains graphical representations of the speed lower bound and the speed upper bound as well as smoothed participant line and reference line speeds during the last trial.
FFR–24	During the “Familiarization trial evaluation” scene, a participant may click the left or right mouse button to switch the scene to the “Camera data saving”.
FFR–25	During the “Camera data saving” scene, the application must stop the current capture, save the current capture, close the current measurement, and disconnect from the motion capture system.
FFR–26	During the “Camera data saving” scene, a participant must see the text with the information about the data saving process on the screen.
FFR–27	During the “Camera data saving” scene, a participant may click on the left or right mouse button to return to the “Capture starting” scene. If the participant performs a certain number of good trials, the scene must switch to the “Familiarization completed” instead of returning to the “Capture starting” scene.
FFR–28	During the “Familiarization completed” scene, a participant must see “Familiarization completed” text at the screen.

Table 5.1: Familiarization block functional requirements.

5.3 Graphical UI to configure the experiments - design

The new application screen that will allow managing participant data and customizing experiment and familiarization blocks parameters will be called a “setup screen”. Before writing the formal list with functional requirements for the new setup screen, participant data structure and customizable experiment and familiarization blocks parameters need to be defined.

Participant data files will be represented as CSV files and will contain headers and values that were a priori chosen and described at the table 5.2 below.

Column number in a csv file	Header	Contents	Can be parsed with type
1	participant_codename	Participant codename, 8 alphanumeric characters.	String
2	age	Participant age.	Int
3	gender	Participant gender: male or female.	String
4	length_of_forearm	Arm length in cm as measured manually by the experimenters.	Double
5	handedness	Right-handed/Left-handed/Ambidextrous.	String
6	sensitivity_to_tactile_test	Value of detected Stoelting test [15], e.g. 0.04 (grams).	Double
7	order_of_the_blocks	One of the 24 possible orders of 4 types of blocks: AT, AM, PT, PM, e.g. AT–>AM–>PT–>PM.	String
8	description	The one short text line.	String

Table 5.2: Participant data file CSV structure.

During the design of the familiarization block, several customizable parameters were a priori preselected to be set by the UI before the familiarization block starts. These parameters are described in Table 5.3 below.

Familiarization block customizable parameter name	**Contents**	**Data type in the application**	**Measurement units**	**Default value**
Speed lower bound	The lower bound of the speed boundaries within which participants need to move their hand during the familiarization block.	Double	cm/s	1.5
Speed upper bound	The upper bound of the speed boundaries within which participants need to move their hand during the familiarization block.	Double	cm/s	4.5
Reference line speed	The speed of the reference line as it grows during the constant speed period.	Double	cm/s	3
Reference line acceleration/deceleration time	The reference line acceleration and deceleration time defines the duration of its acceleration and deceleration periods.	Double	sec	1
Smoothing rate	The size of the moving average applied to participant and reference lines speed data before the familiarization trial evaluation block starts.	Int	–	10
Lines transparency	Slight progressive transparency for upper and lower borders of GUI lines.	Boolean	–	true
Gray zone size	The horizontal size of the borders that represent the maximum length of the reference line during the familiarization active trial.	Double	cm	1.5
Good trials number	The number of good trials that participants must perform to finish the familiarization block.	Int	–	10
Minimum reference line length	The lower bound of the uniform distribution from which the application selects the maximum length of the reference line at the start of the familiarization active trial scene.	Double	cm	10

Maximum reference line length	The upper bound of the uniform distribution from which the application selects the maximum length of the reference line at the start of the familiarization active trial scene.	Double	cm	25
Lines width	The GUI lines width.	Double	cm	2

Table 5.3: Familiarization block customizable parameters.

Following several customizable parameters of the experiments blocks (AT, AM, PM, PT) were a priori preselected to be set by the UI before the familiarization block starts. These parameters are described in Table 5.4 below.

Experiment block customizable parameter name	**Contents**	**Data type in the application**	**Measurement units**	**Default value**
Lines width	The GUI lines width.	Double	cm	2
Trials number	The number of trials that participants must perform to finish an experiment block.	Int	–	10
Judgement line increasing step	The step of changing the judgment line by participants during the experiment judgement scene.	Double	cm	0.325
Initial judgement line min height	The lower bound of the uniform distribution from which the application selects the initial judgement line length at the start of the experiment judgement scene.	Double	cm	5
Initial judgement line max height	The upper bound of the uniform distribution from which the application selects the initial judgement line length at the start of the experiment judgement scene.	Double	cm	25

Table 5.4: Experiment blocks customizable parameters.

To formally describe the requirements for the setup screen, a set of functional requirements has been created and described in Table 5.5.

Code	Functional requirement
SFR–1	After starting the application, the user must see the setup screen, which includes the following components: information about the currently selected participant data, a button labeled “edit” next to the information about the currently selected participant data, a drop-down list with selected and available application blocks (familiarization block, AT, AM, PT, PM), an input form with a set of customizable parameters related to the selected block, a menu bar at the top of the screen containing one item with the text “participant”, the “start the block” button.
SFR–2	The information about the currently selected participant data must consist of headers and values that correspond to the participant data structure defined above. After the application starts, no participant data must be selected by default. If no participant data is selected, participant data headers must be displayed on the setup screen, but any values of information about participants must not be displayed.
SFR–3	If a user clicks the “edit” button, the participant edit dialog must appear above the setup screen, and any interaction with the setup screen must be blocked while the participant edit dialog is visible. A user can click the “edit” button only if participant data is currently selected.
SFR–4	The participant edit dialog contains the “cancel” button and the “save button”, an input form with a set of editable fields that represent the participant data structure and is filled with information about the currently selected participant data.
SFR–5	If a user clicks on the “cancel” button in the participant edit dialog, the dialog must no longer be visible, and any interaction with the setup screen must be unblocked.
SFR–6	If a user clicks on the “save” button in the participant edit dialog, the default file saving dialog defined by the operating system on which the application is running must appear on the screen. Additionally, any interaction with the participant edit dialog must be blocked while the file saving dialog is visible.
SFR–7	If a user clicks the “save” button on the default file saving dialog, the participant information entered in the participant edit dialog must be saved as a file. The file will be saved at the path selected by the user with the name chosen by the user. After saving, both the default file saving dialog and the participant edit dialog must no longer be visible on the screen. The content of the saved file must correspond to the CSV file structure described in Table 5.2.
SFR–8	If a user clicks the cancel button on the default file saving dialog, the default file saving dialog must no longer be visible on the screen.

SFR–9	A user can change the currently selected block by using the drop-down list with selected and available application blocks (familiarization block, AT, AM, PT, PM), and the input form with a set of related customizable parameters must be changed as well.
SFR–10	A user can click on the “participant” element in the menu bar, and then the following elements must appear: “open”, “create”, “create and open”, “exit”.
SFR–11	If a user clicks on the “open” element, the default open file dialog defined by the operating system on which the application is running must appear. After the user chooses a file and clicks on the “open” button, if this file corresponds to the CSV structure for the participant data described above, it must be selected and displayed as the currently selected participant data.
SFR–12	If a user clicks on the “create” element, the participant edit dialog must appear above the setup screen, and any interaction with the setup screen must be blocked until the participant edit dialog is closed. The dialog should contain editable fields that represent the participant data structure, but these fields must not be pre-filled with any information.
SFR–13	If a user clicks on the “create and open” element, the participant edit dialog must appear above the setup screen, and any interaction with the setup screen must be blocked until the participant edit dialog is closed. The dialog should contain editable fields that represent the participant data structure, but these fields must not be pre-filled with any information. After saving the participant data, it should be chosen and displayed as the currently selected participant data.
SFR–14	If a user clicks on the “exit” element then the application must close itself.
SFR–15	If a user clicks on the “start the block” button and the familiarization block is selected in the drop-down list, the application must close the setup screen, open the familiarization screen, and provide the currently selected participant data and the familiarization block customizable parameters to the corresponding classes for the familiarization screen.
SFR–16	If a user clicks on the “start the block” button and an experiment block is selected in the drop-down list, the application must close the setup screen, open the experiment screen, and provide the currently selected participant data and the experiment block customizable parameters to the corresponding classes for the experiment screen.

Table 5.5: Setup screen functional requirements.

5.4 Data logging - design

The application will generate several output files:

1. The output file from Qualisys motion capture system for each trial;
2. The output file that contain information about the experiment block;
3. The output file with detailed information for every trial in the familiarization block;
4. The output file with information about every trial that was performed during an experiment block.

Examples of each of these files are provided in the directory "./cpp/output_data_example" on the project's GitLab repository [11].

The fourth file from the list above will contain nothing more than information described in Tables 5.2 and 5.3, therefore the presentation of its structure within this section is redundant.

The Qualysys output file will be generated by default functions of the Qualysis Track Manager application at the measurement close and will contain information about markers positions and following events that were sent by the tele-touch application during a capture: "3_2_1_scene_started", "active_trial_scene_started", "evaluation_scene_started". These events correspond to moments when the countdown scenes, hand tracking process, or judgment and evaluation scenes start in the application.

All non-Qualisys output data files will be in CSV format, with structures defined in Tables 5.6, 5.7, and 5.8 below.

Sometimes during a trial hand tracking period, a participant may make unexpected back-and-forth movements. In such cases, the distance that the robot performed will be higher than the difference between its start and end positions. To detect trials with such unexpected movements, parameters "robot_initial_coordinates", "robot_final_coordinates", "motion_windows_begin_coordinates" and "motion_windows_end_coordinates" are used in the output file described in the Table 5.7. To clarify the purpose of those parameters, three graphical representations of the trial with unexpected movement during the hand tracking period are depicted in Figures 5.3, 5.4, and 5.5.

Column number in a csv file	**Header**	**Contents**	**Can be parsed with type**
1	participant_codename	Participant codename, at most 8 alphanumeric characters.	String
2	age	Participant age.	Int
3	gender	Participant gender: male or female.	String
4	length_of_forearm	Arm length in cm as measured manually by the experimenters.	Double
5	handedness	Right-handed/Left-handed/Ambidextrous.	String
6	sensitivity_to_tactile_test	Value of detected Stoelting test [15], e.g. 0.04 (grams).	Double
7	order_of_the_blocks	One of the 24 possible orders of 4 types of blocks: AT, AM, PT, PM, e.g. AT–>AM–>PT–>PM.	String
8	description	The one short text line.	String
9	block_type	The chosen block type: AM, AT, PM, or PT.	String
10	condition	Active or passive motion condition derived from a block type.	String
11	judgement_type	Movement or touch judgement type derived from a block type.	String
12	participant_arm_length_cm	The arm length is measured by the Qualisys motion capture system as the difference between the marker on the carpal and the marker on the elbow.	Double

13	total_trials	The total number of trials in this block.	Int
14	trials_done	The total number of completed trials in the experiment block.	Int
15	no_of_cameras	The number of motion capture system cameras that were used during the experiment block.	Int
16	no_of_markers	The number of different markers that were detected by the motion capture system during the experiment block.	Int
17	cameras_frequency	The camera's recording frequency in Hz that was used for recording during the experiment block.	Int
18	no_of_frames	The number of frames that was recorded during the experiment block.	Int
19	lines_width_cm	The customizable parameter of GUI line width.	Double
20	initial_judgement_line _max_height_cm	The upper bound of the uniform distribution from which the application selects the initial judgement line length at the start of the experiment judgement scene.	Double
21	initial_judgement_line _min_height_cm	The lower bound of the uniform distribution from which the application selects the initial judgement line length at the start of the experiment judgement scene.	Double
22	judgement_line_increasing _step_cm	The step of changing the judgment line by participants during the experiment judgement scene.	Double
23	block_end_datetime	The date and time of the block ending.	String

Table 5.6: Experiment block output file CSV structure.

Column number in a file	**Column name**	**Contents**	**Can be parsed with type**
1	participant_codename	Participant codename, at most 8 alphanumeric characters.	String
2	age	Participant age.	Int
3	gender	Participant gender: male or female.	String
4	length_of_forearm	Arm length in cm as measured manually by the experimenters.	Double
5	handedness	Right-handed/Left-handed/Ambidextrous.	String
6	sensitivity_to_tactile_test	Value of detected Stoelting test [15], e.g. 0.04 (grams).	Double
7	order_of_the_blocks	One of the 24 possible orders of 4 types of blocks: AT, AM, PT, PM, e.g. AM–AT–PT–PM.	String
8	description	The one short text line.	String
9	block_type	The chosen block type: AM, AT, PM, or PT	String
10	trial_number	The number of trials in the sequence of all trials in the current block.	Int
11	total_trial_number	The total number of trials in the current block.	Int
12	gain	The gain factor with which the robot repeats the participant's movements.	Double
13	trial_start_datetime	The date and time when the trial started.	String
14	robot_participant_sync _datetime	The date and time when the robot started to repeat the participant's movements.	String

15	robot_participant_sync _frame	The Qualysis motion capture system frame when the robot started to repeat the participant's movements.	Int
16	judgement_start_datetime	The date and time when the scene was switched to the judgement scene.	String
17	judgement_start_frame	The Qualysis motion capture system frame when the scene was switched to the judgement scene.	Int
18	judgement_length_cm	The judgement line length that was set by a participant during the judgement scene.	Double
19	robot_initial_coordinates	The robot coordinates when the robot started to repeat the participant's movements.	String with coordinates in the following format: "x;y;z"
20	robot_final_coordinates	The robot coordinates when the robot ended the repetition of the participant's movements.	String with coordinates in the following format: "x;y;z".
21	motion_windows_begin _coordinates	Minimal coordinates that robot reached in the current trial.	String with coordinates in the following format: "x;y;z".

22	motion_windows_end _coordinates	Maximal coordinates that robot reached in the current trial.	String with coordinates in the following format: “x;y;z”.
23	trial_end_datetime	The date and time of trial ending.	String

Table 5.7: Experiment trials information output file CSV structure.

Column number in a file	**Column name**	**Contents**	**Can be parsed with type**
1	timestamp	The date and time at the moment when the image redraws on the screen.	String
2	frame_number	The last received Qualysis motion capture system frame number at the moment when the image redraws on the screen.	Int
3	participant	Participant codename, at most 8 alphanumeric characters.	String
4	trial_number	The number of the current trial.	Int
5	good_trials	The number of good trials that a participant already made.	Int
6	participant_line_height _cm	The participant line height at the moment when the image redraws on the screen.	Double
7	reference_line_height _cm	The reference line height at the moment when the image redraws on the screen.	Double
8	participant_line_speed _cm_s	The current participant line speed that computes the difference between previous participant line height and the current participant line height divided by difference between current time and last image redraw time.	Double

9	smoothed_participant_line _speed_cm_s	The current smoothed participant line speed.	Double
10	reference_line _speed_cm_s	The current reference line speed that computes the difference between previous reference line height and the current reference line height divided by difference between current time and last image redraw time.	Double
11	smoothed_reference_line _speed_cm_s	The current smoothed reference line speed.	Double
12	reference_line_final_length _cm	The final reference line length at the current trial.	Double

Table 5.8: Familiarization trial information output file CSV structure.

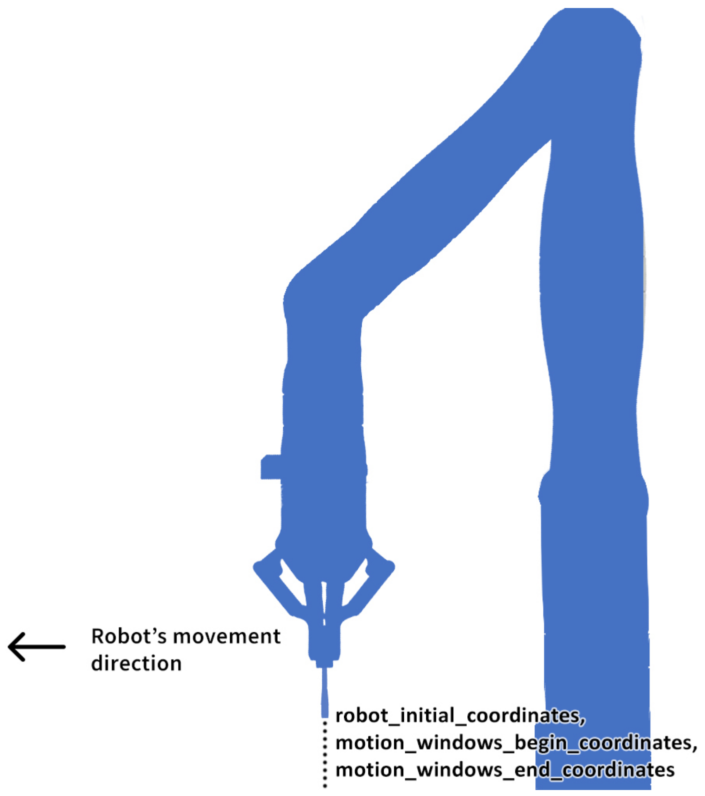

Robot's movement direction
robot_initial_coordinates,
motion_windows_begin_coordinates,
motion_windows_end_coordinates

Figure 5.3: Example of the robot movement start position.

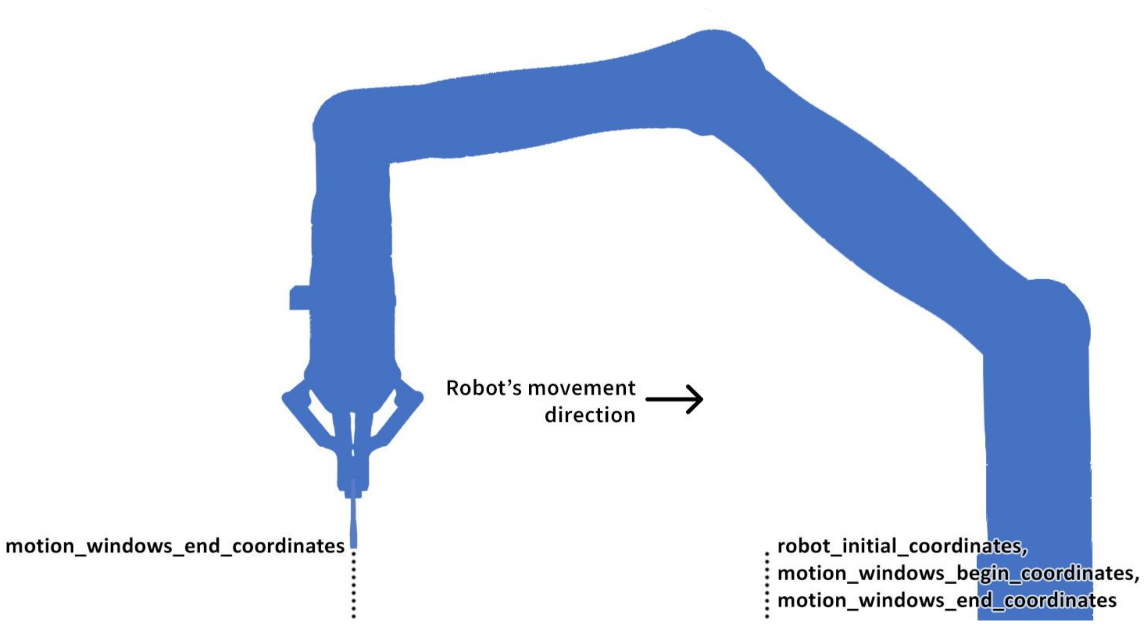

Robot's movement direction
motion_windows_end_coordinates
robot_initial_coordinates,
motion_windows_begin_coordinates,
motion_windows_end_coordinates

Figure 5.4: Example of the robot movement intermediate position.

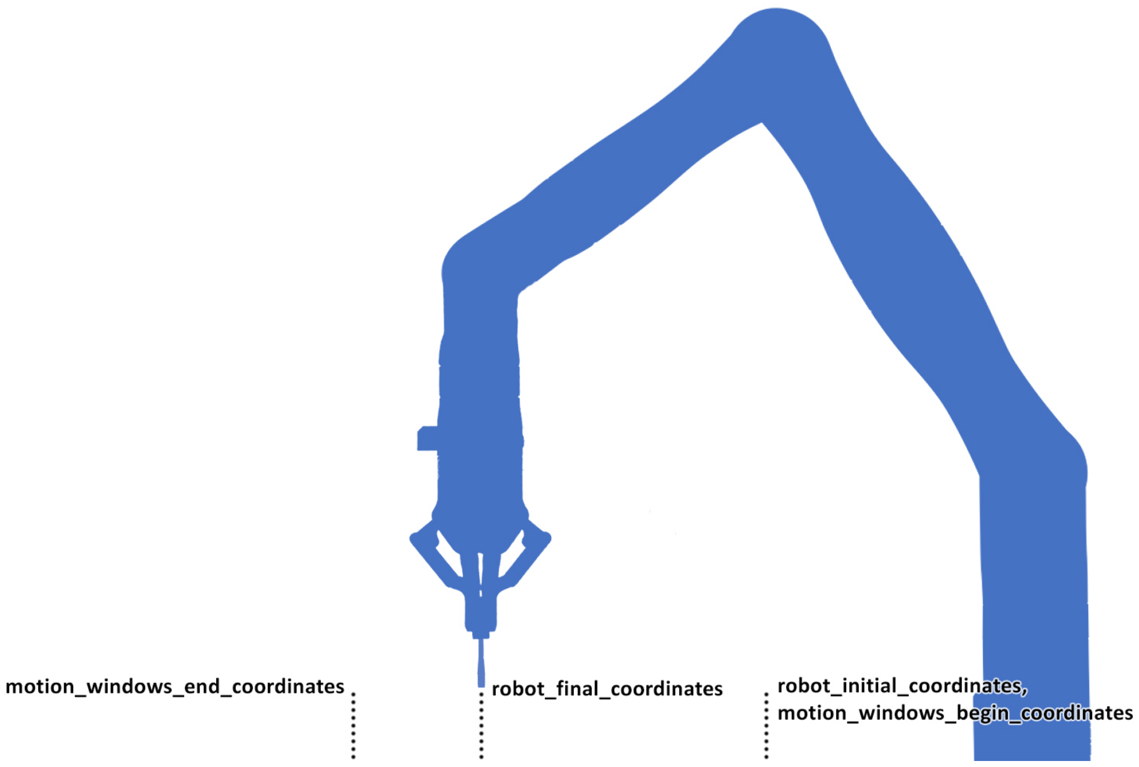

motion_windows_end_coordinates
robot_final_coordinates
robot_initial_coordinates,
motion_windows_begin_coordinates

Figure 5.5: Example of the robot movement end position.

To formally describe the requirements for the logging of the data for evaluation of the experiment process, a set of functional requirements has been created and described in Table 5.9.

LFR–1	At the start of the familiarization block trial, the familiarization trial file with structure defined in the Table 5.8 must be created.
LFR–2	At the end of the familiarization block, the output file with information about every trial that was performed during the familiarization block must be created.
LFR–3	During the familiarization active trial scene, when the image redraws on the screen, the data line containing values that correspond to the structure defined in Table 5.8 must be added to the familiarization trial file with the trial data.
LFR–4	At the start and the end of the experiment block, the experiment block info output file must be created and filled with two CSV lines that must correspond to the header and values described in the table 5.6.
LFR–5	At the start of the experiment block the experiment trials output file must be created and filled with headers that correspond to structure defined in the Table 5.7.
LFR–6	At the end of the experiment trial the data line with the trial information that corresponds to structure defined in the Table 5.7 must be added to the experiment trial output file.
LFR–7	Before every familiarization or experiment trial starts the application must open a new measurement and start a new capturing in the Qualisys motion capture system.
LFR–8	After every familiarization or experiment trial starts the application must stop the current capture, save the current capture, close the current measurement that must cause the motion capture output file creation in the Qualisys motion capture system.
LFR–9	If the countdown scene has started, the application must send the event "3_2_1_scene_started" to the Qualysis motion capture system.
LFR–10	If the familiarization active trial scene or active trial scene (from an experiment block) has started, the application must send the event "active_trial_scene_started" to the Qualysis motion capture system.
LFR–11	If the familiarization trial evaluation or judgement scene (from an experiment block) has started, the application must send the event "evaluation_scene_started" to the Qualysis motion capture system.

Table 5.9: Data logging functional requirements.

5.5 2D extension - design

The legacy application contains logic that allows participants to control the robot in 1D space. However, for future experiments, the logic for controlling the robot in 2D space is required. Since the specific design of these future experiments is currently unknown, it is necessary to implement the basic logic for controlling the robot in 2D space to establish a foundation for future experiments.

The 2D controlling logic must not interfere with the familiarization and experimental application blocks, therefore this logic will be used only in the new application block that will be called the “Test 2D” block. The “Test 2D” block will include the following scenes: Capture starting, Click to start tracking, Movement is tracking, and the Error scenes. At any given moment, only one scene may be active. Possible scene transitions are illustrated in Figure 5.6.

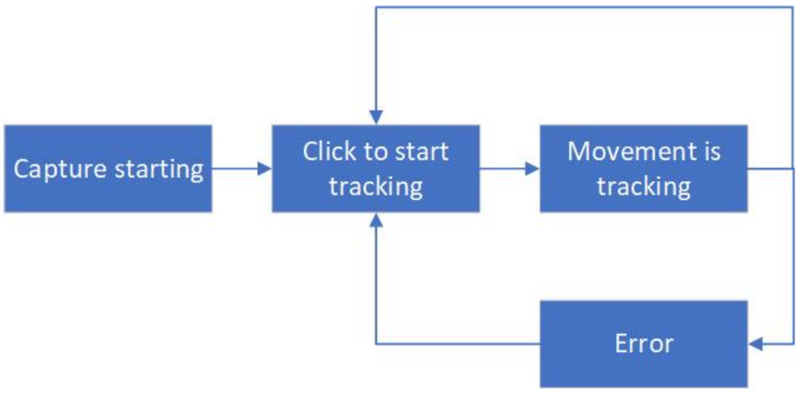


Figure 5.6: Test 2D block’s scene transitions.

The safety of 2D robot controlling must be ensured by using the copy of the legacy application functions that restricts robot movements within a defined boundary box and check that the robot’s speed is below a specified maximum speed value. If those functions detect that the application is attempting to exceed the boundary box limits or move the robot at a speed higher than the maximum, the scene must switch to the error scene, and the robot must stop hand tracking and return to the home position.

The logic that needs to be implemented is described as a set of functional requirements provided in the Table 5.10 below.

2DFR–1	After the “Test 2d” block starts, the application must connect to the robot and the “Capture started” scene must start.
2DFR–2	After the “Capture starting” scene start the application must send the command to the robot to start moving to the robot’s home position.
2DFR–3	During the “Capture starting” scene, a participant must see on the screen the text “Capture starting”.
2DFR–4	After the “Capture starting” scene starts, the application must connect to the motion capture system, open a new measurement, start a new capturing and wait until the robot reaches the robot’s home position.
2DFR–5	During the “Capture starting” scene, if the application is connected to the motion capture system and a new capture is started and the robot reached the robot’s home position, then the scene must switch to the “Click to start tracking”.
2DFR–6	During the “Click to start tracking” scene, a participant must see on the screen the text “Click to start tracking”.
2DFR–7	During the “Click to start tracking” scene, a participant may click on the left or right mouse button to go to the “Movement is tracking” scene.
2DFR–8	During the “Movement is tracking” scene, the robot must start to repeat the participant’s left hand movement in 2D with movements in the x and y axis.
2DFR–9	If during the “Movement is tracking” scene, the application will try to send coordinates that will result in robot movement outside the defined boundary box to repeat the participant’s left hand movement, then the scene must switch to the “Error” scene and those coordinates must not be sended.
2DFR–10	If during the “Movement is tracking” scene, the application will try to send coordinates that will result in robot movement with higher than defined maximum speed to repeat the participant’s left hand movement, then the scene must switch to the “Error” scene and those coordinates must not be sended.
2DFR–11	During the “Movement is tracking” scene, a participant may click on the left or right mouse button to go to the “Click to start tracking” scene.

2DFR–12	At the end of the “Movement is tracking” scene, the application must send commands to the robot to stop the hand tracking and start moving to the robot’s home position.
2DFR–13	During the “Error” scene, a participant must see on the screen the text “Error happened. The robot tried to move outside of the boundary box or with higher than maximum speed.”.
2DFR–14	During the “Error” scene, a participant may click on the left or right mouse button to go to the “Click to start tracking” scene.

Table 5.10: 2D block functional requirements.

Chapter 6

Software application implementation

During my work on the diploma thesis, I implemented the logic based on the specifications provided in Chapter 5. This chapter contains a description of the results of the implementation process and describes all implemented logic, except for small utility classes that do not contain complex logic.

In every section of this chapter, several main interfaces and classes that implement the logic related to a section were identified and described by informal class descriptions. These classes are also represented in the UML class diagram [16] and shown in a project folder structure generated by "tree" command [17], to demonstrate the implementation of recommended design principles and the architecture described in Sections 4.1 and 5.1. In the UML class diagrams, classes that were built as a result of the refactoring process during the implementation of the new architecture, but whose main logic remains the same as in the legacy application or is mostly based on the legacy application logic, have an aqua background. Other classes, which were written from scratch, have a white-gray background. Methods arguments, class data members, and multiplicity denotations were removed from UML class diagrams to reduce and simplify them. From the information provided by the UML class diagrams and directory structures, it can be concluded that every implemented class is represented by its own `.cpp` and `.h` files with names that reflect their purpose. These files are grouped by directories with appropriate names. It can also be observed that interfaces and classes are application components, and their names and methods indicate their semantic purpose.

Screenshots of different scenes in the working application are provided as showcases in every section with GUI logic. Examples of output files are provided as showcases in the 6.3 Section. All source code can be found in the tele-touch GitLab project's main branch [11] or in the attachment for the diploma thesis.

6.1 Familiarization block - implementation

The several interfaces and the one static class have an important role in familiarization block implementation. Their descriptions is provided below:

- FamiliarizationView – the view that defines functions required for drawing the familiarization block scenes that must be implemented by the view implementation.

- Service – the empty marker interface to identify services.
- SingletonServiceProvider – the special static class that contains logic for managing singleton services. A singleton service is a service that can have only one instance during the application's runtime, which can be useful for avoiding unnecessary interference while using APIs.

The familiarization block logic contains many different classes, but following classes were identified as the core and most important classes of the familiarization block:

- RobotWorker – the application worker that holds all the logic for robot communication via the Kortex API [18]. The worker operates in a separate thread created by the `RobotService` and overloads the function call operator "`operator()`". It receives and sends data using shared memory. To receive commands, the `RobotWorker`'s shared memory has a command list which is essentially a queue of commands of a certain type that a robot can perform, defined in the `ROBOT_INPUT_COMMAND_TYPE` enum class. Only the `RobotService` and `RobotWorker` have access to their shared memory.
- PositionComputer – the application utility that contains the main critical methods for computing robot trajectories during trials and ensuring safety during hand tracking.
- RobotService – the singleton service that is provided by request of the `Singleton ServiceProvider`. It contains service methods to operate the robot. All method names represent their semantics and allow any presenter to easily use the `RobotService` for robot controlling by calling service methods without worrying about shared data and implementation details.
- CameraWorker – the application worker that holds all the logic for robot communication via the Qualysis API [10]. Similar to the `RobotWorker`, it operates in a separate thread created by the `CameraService` and overloads the function call operator "`operator()`". It receives and sends data using shared memory, with a command list for receiving commands defined in the `CAMERA_INPUT_COMMAND_TYPE` enum class. Only the `CameraService` and `CameraWorker` have access to their shared memory.
- CameraService – similar to the `RobotService`, this singleton service that is provided by request of the `SingletonServiceProvider`. It contains service methods to operate the Qualisys motion capture system. Method names represent their semantics and allow any presenter to easily control the motion capture system without worrying about shared data and implementation details.
- FamiliarizationBlockLoggerService – the application services that contains methods for logging information during the familiarization block. The structure of this information was shown in Tables 5.8, 5.2 and 5.3.
- FamiliarizationViewPresenter – This application presenter implements the logic for reacting to events that happen in the view. It contains methods for orchestrating robot and camera services, familiarization block data processing, and computing new UI element states during screen updates.

- FamiliarizationFrame – the `FamiliarizationView` implementation that contains methods for drawing GUI elements during the familiarization block running. It uses `RenderTimer` to update the screen at a frequency of 100 Hz (if the screen supports it), `CustomDrawer` and `GridDrawer` utilities to draw GUI components, `ScreenSetting` for computing the real size of familiarization lines that represent participant's and reference's lengths of movement, and `wxSpeedEvaluationGraphCtl` to draw a graph with participant's and reference's lines speed during the familiarization trial.
- CustomDrawer – the view graphical component contains logic for drawing GUI objects onto `wxDC` ("device context") [19] component that is used for drawing custom GUI elements in the wxWidgets library [20], which is the application's basic graphical library.
- GridDrawer – the view graphical component that inherits from `CustomDrawer` and includes additional logic for aligning elements by an abstract grid with given sizes onto the `wxDC` component.
- wxSpeedEvaluationGraph and wxSpeedEvaluationGraphCtl are view graphical components that contain logic for drawing graphs in the application. These classes inherit from the wxCharts library [21] classes and modify and improve some of their functions. This was necessary because the library received its last update two years ago, and some parts of its logic work in unexpected ways or don't work at all. The library was chosen because it provides the ability to draw various graphs that are best suited for rendering speed graphs and look better compared to other free libraries.
- RenderTimer – the view component that contains logic for generating update events for a given view with a given frequency.
- ScreenSettings – the application utility that provides information about the screen parameters on which the application is displayed in order to draw GUI elements with sizes defined in metric units.

The UML class diagram for interfaces and classes described above, along with all the methods they contain, is shown in Figures 6.1, 6.2, and 6.3.

The directory structure diagram, presented as a directory tree with familiarization block classes as leaves, is shown in Figure 6.4.

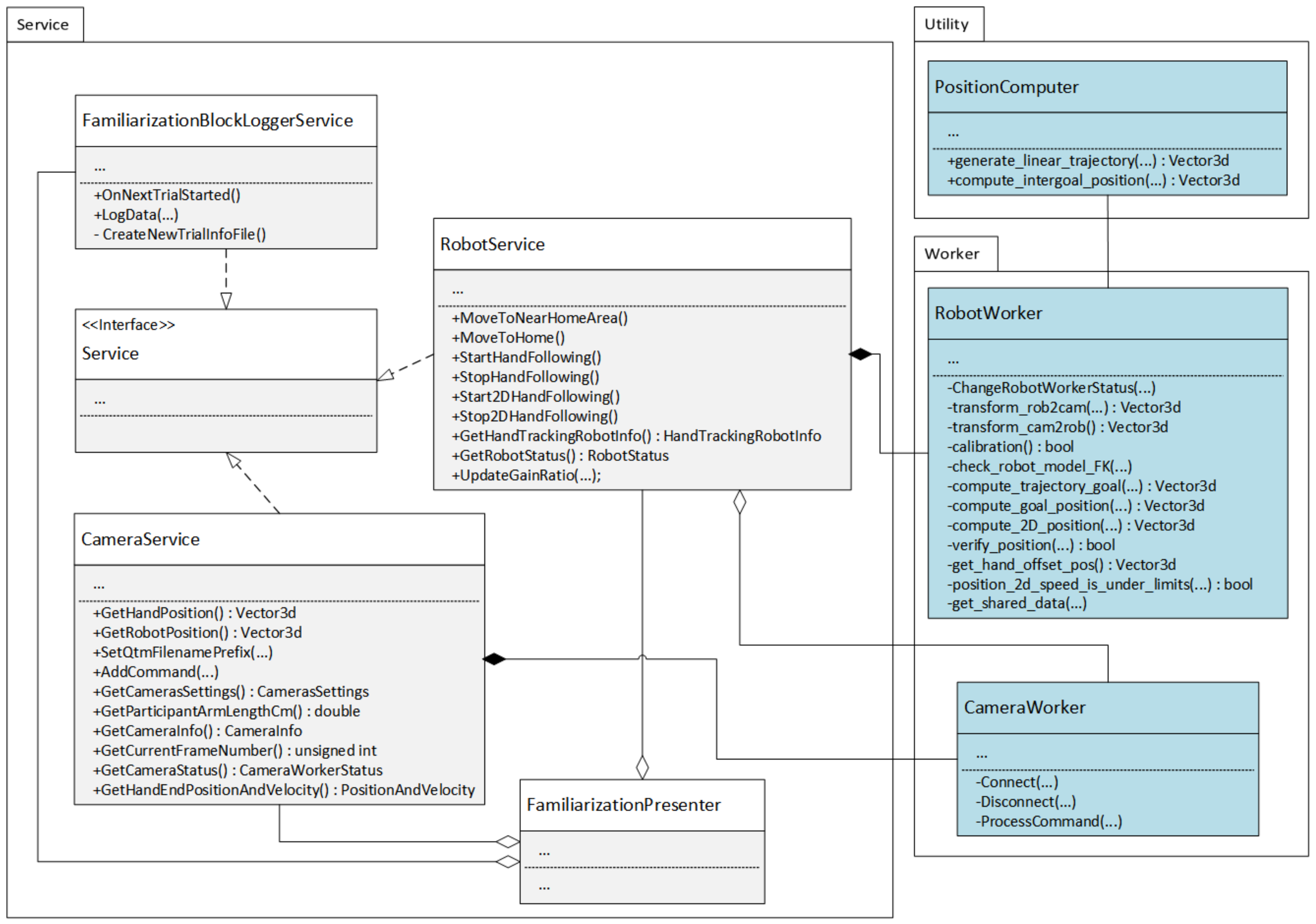

Service
FamiliarizationBlockLoggerService
...
+OnNextTrialStarted()
+LogData(...)
- CreateNewTrialInfoFile()
<<Interface>>
Service
...
RobotService
...
+MoveToNearHomeArea()
+MoveToHome()
+StartHandFollowing()
+StopHandFollowing()
+Start2DHandFollowing()
+Stop2DHandFollowing()
+GetHandTrackingRobotInfo() : HandTrackingRobotInfo
+GetRobotStatus() : RobotStatus
+UpdateGainRatio(...);
CameraService
...
+GetHandPosition() : Vector3d
+GetRobotPosition() : Vector3d
+SetQtmFilenamePrefix(...)
+AddCommand(...)
+GetCamerasSettings() : CamerasSettings
+GetParticipantArmLengthCm() : double
+GetCameraInfo() : CameraInfo
+GetCurrentFrameNumber() : unsigned int
+GetCameraStatus() : CameraWorkerStatus
+GetHandEndPositionAndVelocity() : PositionAndVelocity
FamiliarizationPresenter
...
...
Utility
PositionComputer
...
+generate_linear_trajectory(...) : Vector3d
+compute_intergoal_position(...) : Vector3d
Worker
RobotWorker
...
-ChangeRobotWorkerStatus(...)
-transform_rob2cam(...) : Vector3d
-transform_cam2rob() : Vector3d
-calibration() : bool
-check_robot_model_FK(...)
-compute_trajectory_goal(...) : Vector3d
-compute_goal_position(...) : Vector3d
-compute_2D_position(...) : Vector3d
-verify_position(...) : bool
-get_hand_offset_pos() : Vector3d
-position_2d_speed_is_under_limits(...) : bool
-get_shared_data(...)
CameraWorker
...
-Connect(...)
-Disconnect(...)
-ProcessCommand(...)

Figure 6.1: Familiarization block class diagram. Services, workers and utility.

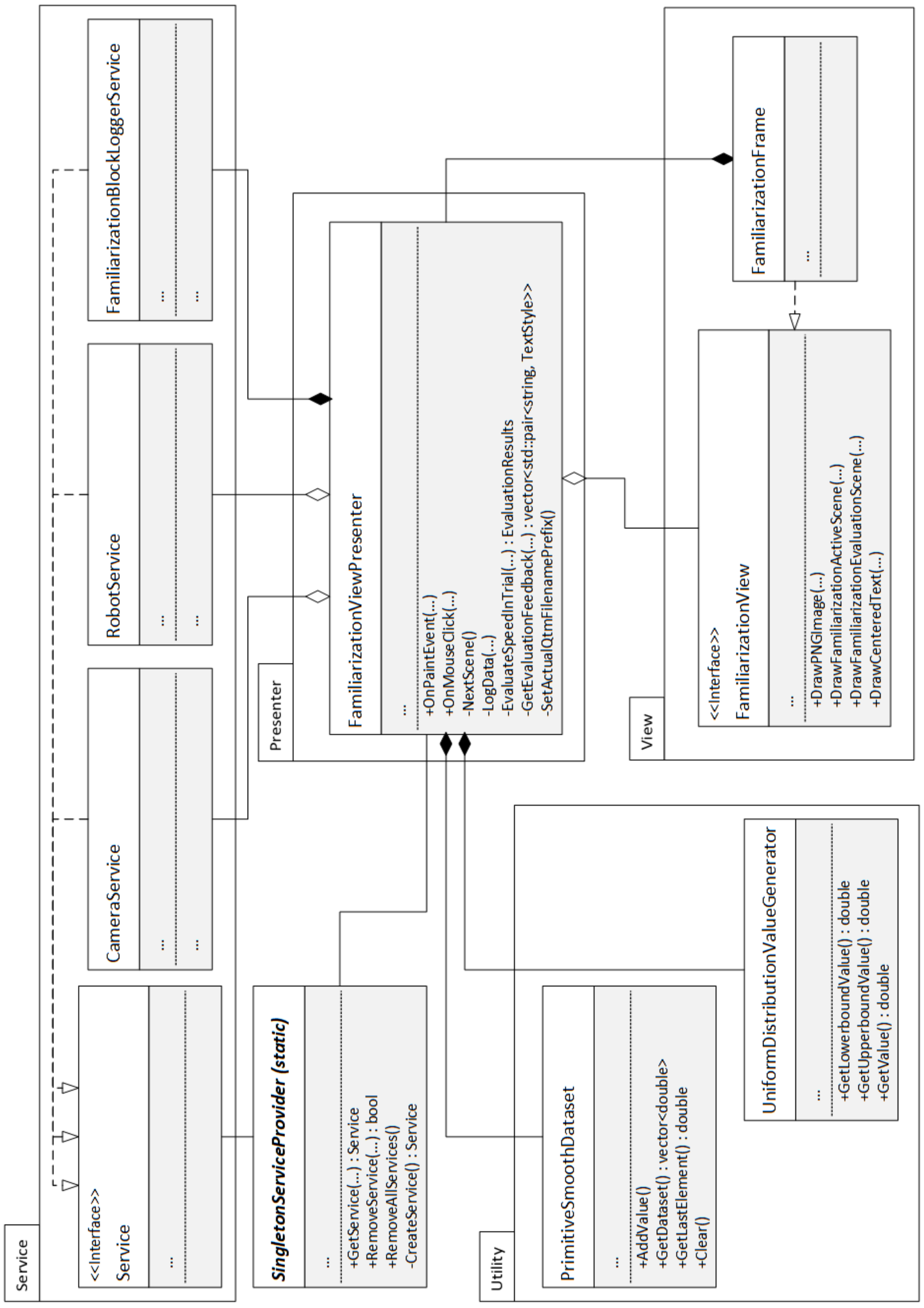

Service
<<Interface>>
Service
...
FamiliarizationBlockLoggerService
RobotService
CameraService
SingletonServiceProvider (static)
+GetService(...) : Service
+RemoveService(...) : bool
+RemoveAllServices()
-CreateService() : Service
Presenter
FamiliarizationViewPresenter
+OnPaintEvent(...)
+OnMouseClick(...)
-NextScene()
-LogData(...)
-EvaluateSpeedInTrial(...) : EvaluationResults
-GetEvaluationFeedback(...) : vector<std::pair<string, TextStyle>>
-SetActualQtmFilenamePrefix()
View
FamiliarizationFrame
<<Interface>>
FamiliarizationView
+DrawPNGImage(...)
+DrawFamiliarizationActiveScene(...)
+DrawFamiliarizationEvaluationScene(...)
+DrawCenteredText(...)
Utility
PrimitiveSmoothDataset
+AddValue()
+GetDataset() : vector<double>
+GetLastElement() : double
+Clear()
UniformDistributionValueGenerator
+GetLowerboundValue() : double
+GetUpperboundValue() : double
+GetValue() : double

Figure 6.2: Familiarization block class diagram. Services, utilities, presenter and view.

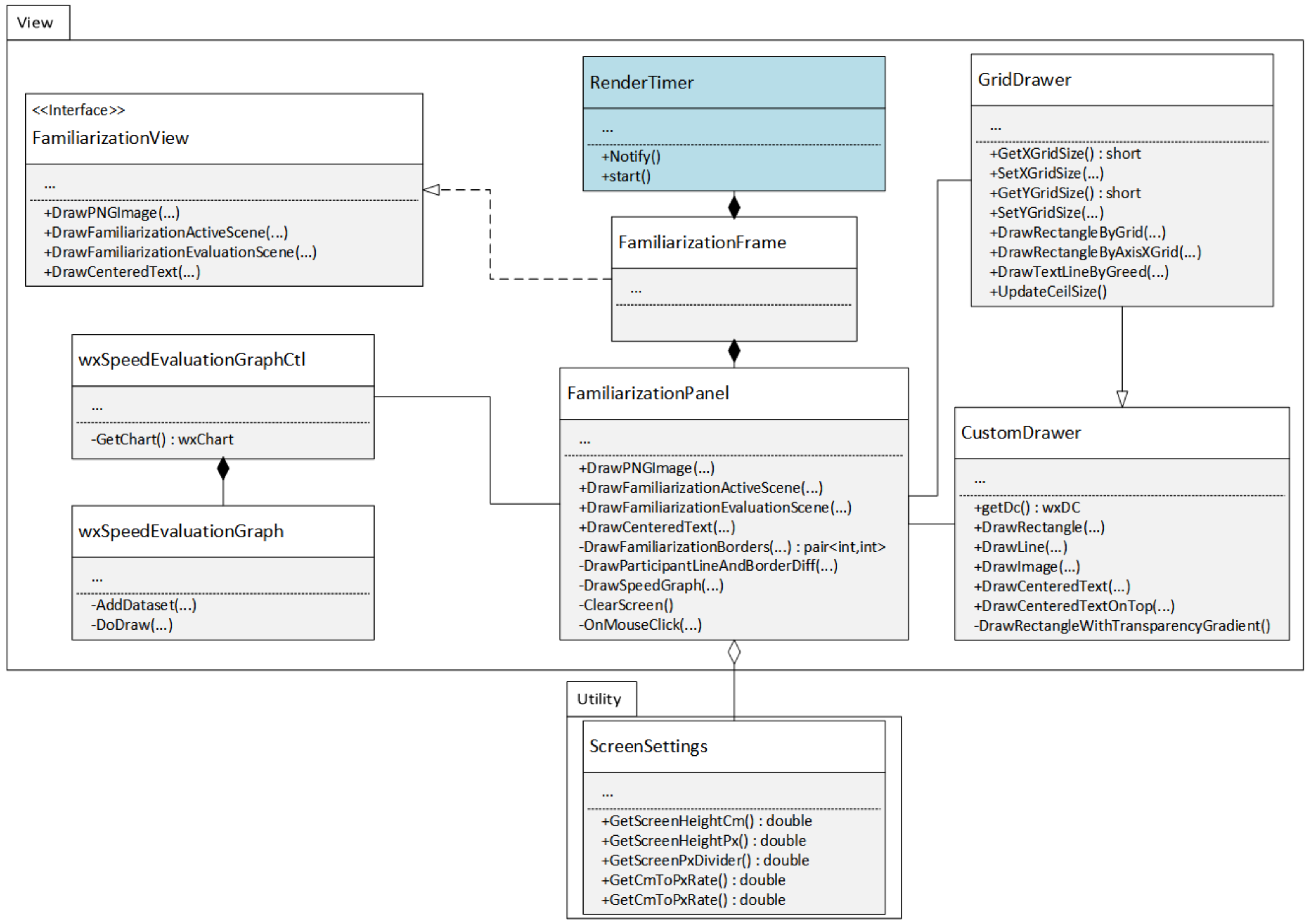
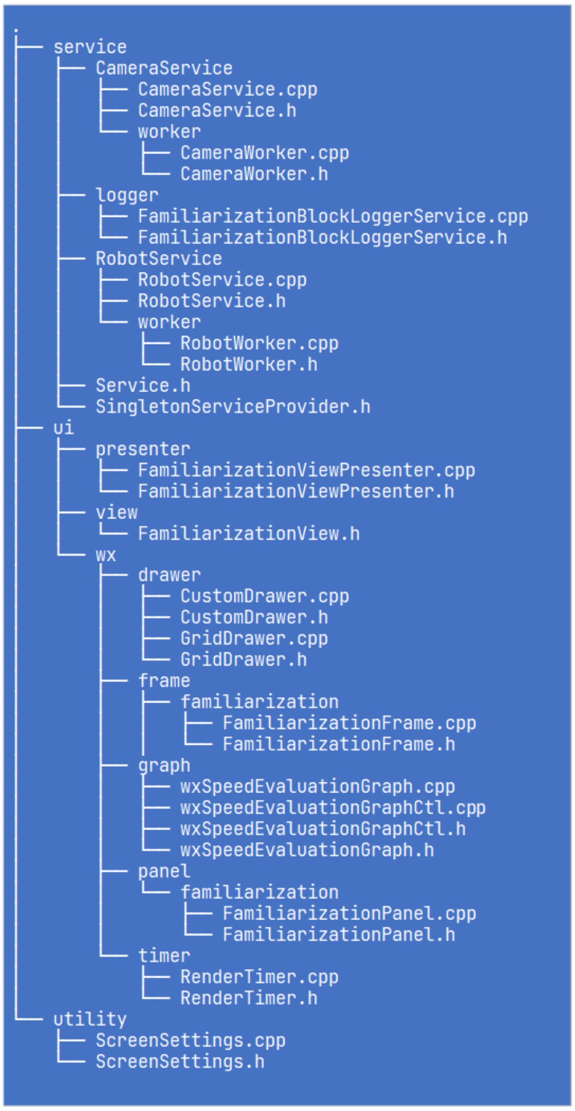


Figure 6.3: Familiarization block class diagram. View, it graphical components, utility.

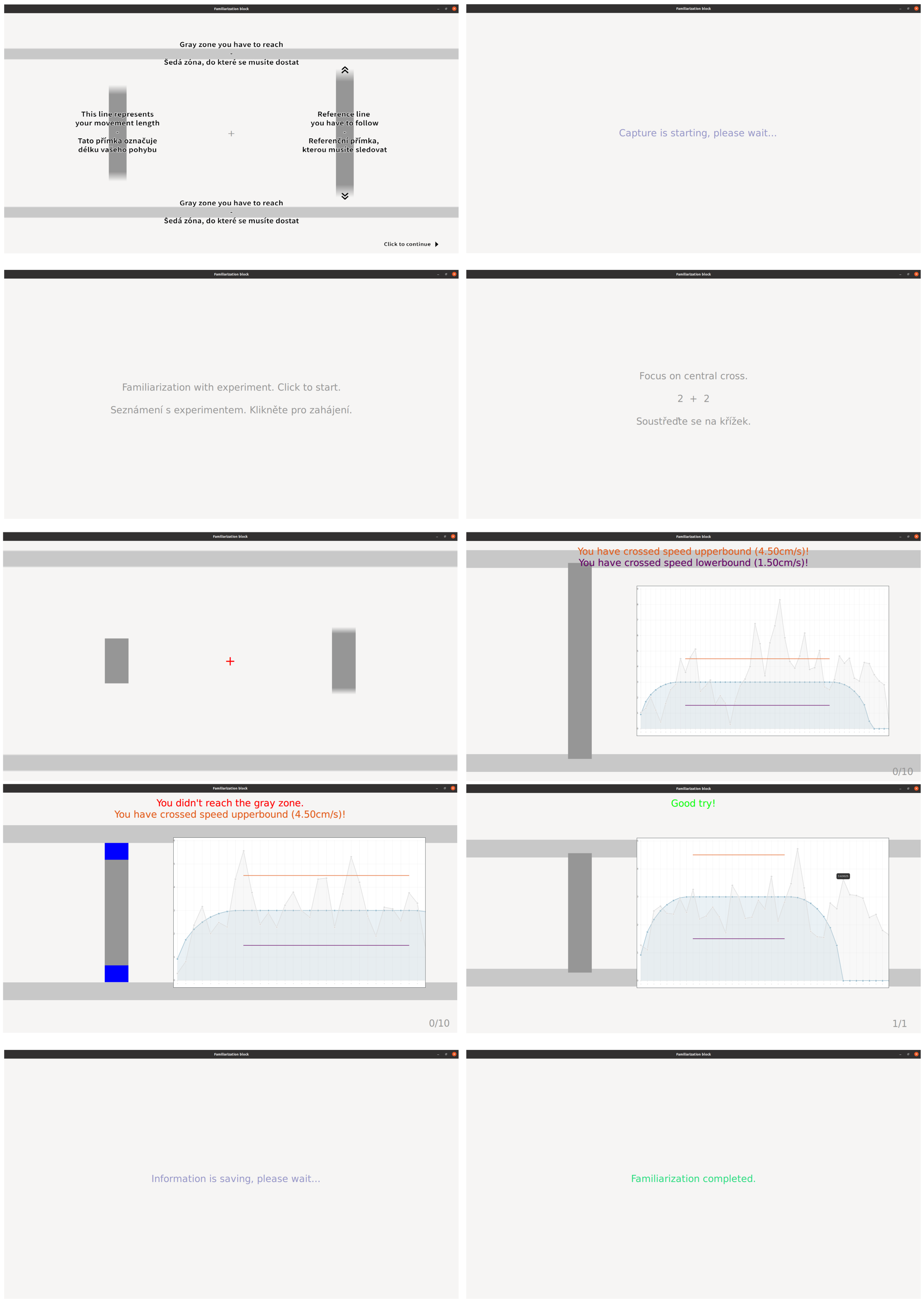


```
.
├── service
│   ├── CameraService
│   │   ├── CameraService.cpp
│   │   ├── CameraService.h
│   │   └── worker
│   │       ├── CameraWorker.cpp
│   │       └── CameraWorker.h
│   ├── logger
│   │   ├── FamiliarizationBlockLoggerService.cpp
│   │   └── FamiliarizationBlockLoggerService.h
│   ├── RobotService
│   │   ├── RobotService.cpp
│   │   ├── RobotService.h
│   │   └── worker
│   │       ├── RobotWorker.cpp
│   │       └── RobotWorker.h
│   ├── Service.h
│   └── SingletonServiceProvider.h
├── ui
│   ├── presenter
│   │   ├── FamiliarizationViewPresenter.cpp
│   │   └── FamiliarizationViewPresenter.h
│   ├── view
│   │   └── FamiliarizationView.h
│   └── wx
│       ├── drawer
│       │   ├── CustomDrawer.cpp
│       │   ├── CustomDrawer.h
│       │   ├── GridDrawer.cpp
│       │   └── GridDrawer.h
│       ├── frame
│       │   └── familiarization
│       │       ├── FamiliarizationFrame.cpp
│       │       └── FamiliarizationFrame.h
│       ├── graph
│       │   ├── wxSpeedEvaluationGraph.cpp
│       │   ├── wxSpeedEvaluationGraphCtl.cpp
│       │   ├── wxSpeedEvaluationGraphCtl.h
│       │   └── wxSpeedEvaluationGraph.h
│       ├── panel
│       │   └── familiarization
│       │       ├── FamiliarizationPanel.cpp
│       │       └── FamiliarizationPanel.h
│       └── timer
│           ├── RenderTimer.cpp
│           └── RenderTimer.h
└── utility
    ├── ScreenSettings.cpp
    └── ScreenSettings.h
```

Figure 6.4: Familiarization block directory structure.

Implemented scenes from the Figure 5.2 shown in the Figure 6.5 from left to right and top to bottom in the following order: UI familiarization, capture starting, block information, 321+ countdown, familiarization active trial, familiarization trial evaluation (with information about crossing the speed lower bound and speed upper bound), familiarization trial evaluation (with information about crossing the speed upper bound and not reaching the border), familiarization trial evaluation (with good trial example), camera data saving, familiarization completed.

Based on the application source code on GitLab [11] and the example of the familiarization block running shown the Figure 6.5, it can be concluded that all the functional requirements described in Table 5.1 have been fulfilled.


Familiarization block

Gray zone you have to reach
-
Šedá zóna, do které se musíte dostat

This line represents
your movement length
-
Tato přímka označuje
délku vašeho pohybu

Reference line
you have to follow
-
Referenční přímka,
kterou musíte sledovat

Gray zone you have to reach
-
Šedá zóna, do které se musíte dostat

Click to continue

Capture is starting, please wait...

Familiarization with experiment. Click to start.

Seznámení s experimentem. Klikněte pro zahájení.

Focus on central cross.

2 + 2

Soustřeďte se na křížek.

You have crossed speed upperbound (4.50cm/s)!
You have crossed speed lowerbound (1.50cm/s)!

0/10

You didn't reach the gray zone.
You have crossed speed upperbound (4.50cm/s)!

0/10

Good try!

1/1

Information is saving, please wait...

Familiarization completed.

Figure 6.5: Familiarization block scenes showcase.

6.2 Graphical UI to configure the experiments - implementation

During the implementation of the graphical UI for configuring experiments, several application components were created. Some of them were already described in the previous section, and descriptions of others are provided below:

- ParticipantService – the singleton service that contains methods for opening, saving, and editing participant information, with the structure defined in Table 5.2 in ROM.
- SetupViewPresenter – the application presenter that processes and reacts to information received as a result of participant info and customizable parameters management events that occur in the view. It uses `ParticipantService` to save information on ROM.
- SetupView – the view that defines functions required to draw the graphical UI for configuring experiments that must be implemented by the view implementation.
- SetupFrame – the view implementation that uses `ParticipantInfoPanel`, `ExperimentControlPanel` and `EditParticipantDialog` view components to draw and operate the GUI for the setup screen.
- EditParticipantDialog – the view component that implements logic for showing the input form as a dialog for editing and creating participant information.
- ParticipantInfoPanel – the view component that implements logic for displaying participant information.
- ExperimentControlPanel – the view component that implements logic for block choosing and for displaying and editing customizable parameters for experiment and familiarization blocks.

The main classes of the graphical UI for configuring experiments are shown in Figures 6.6 and 6.7. The directory tree structure for these classes is shown in Figure 6.8.

The example of setup screen different states during the application running is shown on Figures 6.9, 6.10, 6.11, 6.12.

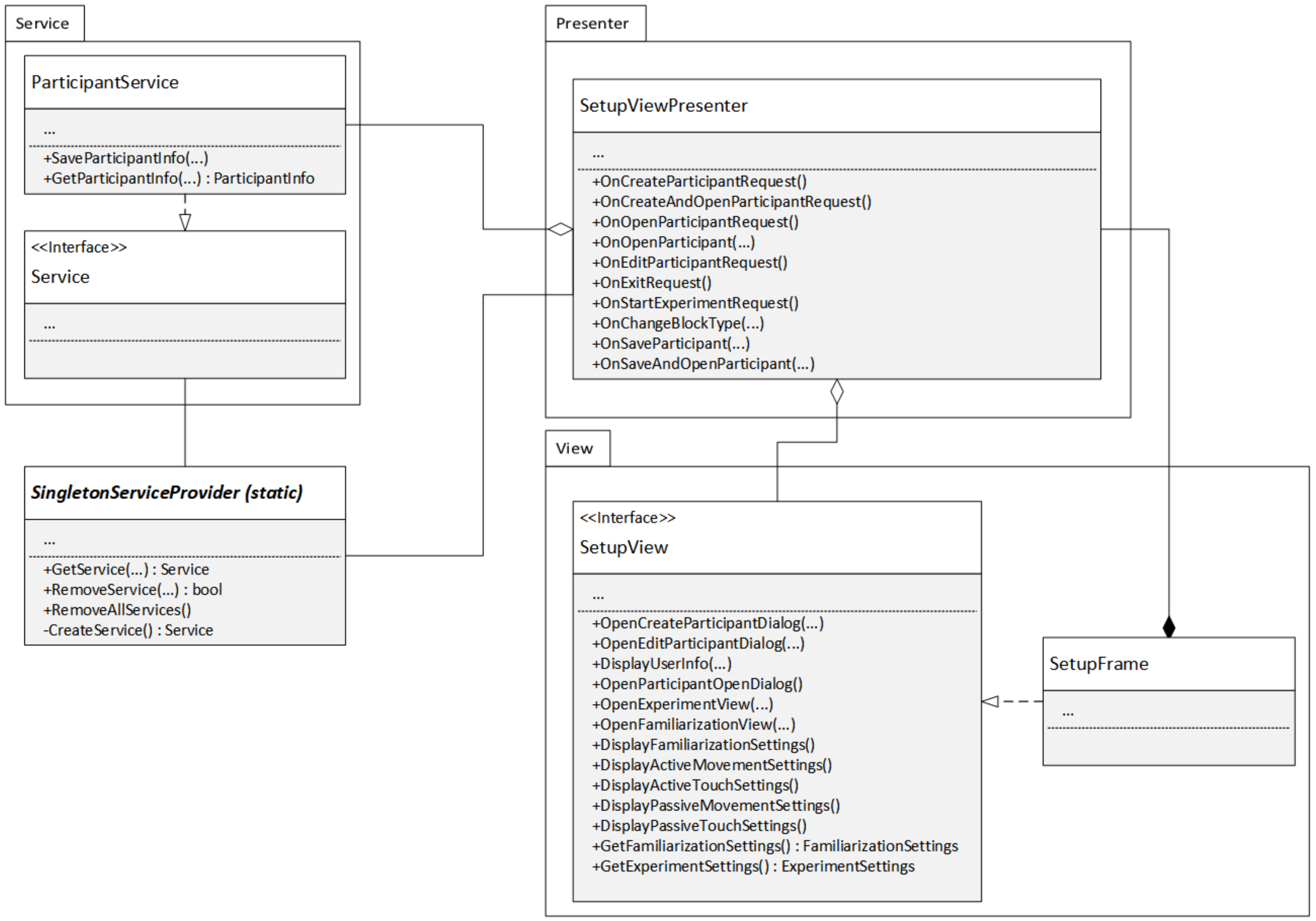

Service
ParticipantService
...
+SaveParticipantInfo(...)
+GetParticipantInfo(...) : ParticipantInfo
<<Interface>>
Service
...
SingletonServiceProvider (static)
...
+GetService(...) : Service
+RemoveService(...) : bool
+RemoveAllServices()
-CreateService() : Service
Presenter
SetupViewPresenter
...
+OnCreateParticipantRequest()
+OnCreateAndOpenParticipantRequest()
+OnOpenParticipantRequest()
+OnOpenParticipant(...)
+OnEditParticipantRequest()
+OnExitRequest()
+OnStartExperimentRequest()
+OnChangeBlockType(...)
+OnSaveParticipant(...)
+OnSaveAndOpenParticipant(...)
View
<<Interface>>
SetupView
...
+OpenCreateParticipantDialog(...)
+OpenEditParticipantDialog(...)
+DisplayUserInfo(...)
+OpenParticipantOpenDialog()
+OpenExperimentView(...)
+OpenFamiliarizationView(...)
+DisplayFamiliarizationSettings()
+DisplayActiveMovementSettings()
+DisplayActiveTouchSettings()
+DisplayPassiveMovementSettings()
+DisplayPassiveTouchSettings()
+GetFamiliarizationSettings() : FamiliarizationSettings
+GetExperimentSettings() : ExperimentSettings
SetupFrame
...

Figure 6.6: Graphical UI for configure experiments class diagram. Services, presenter and view.

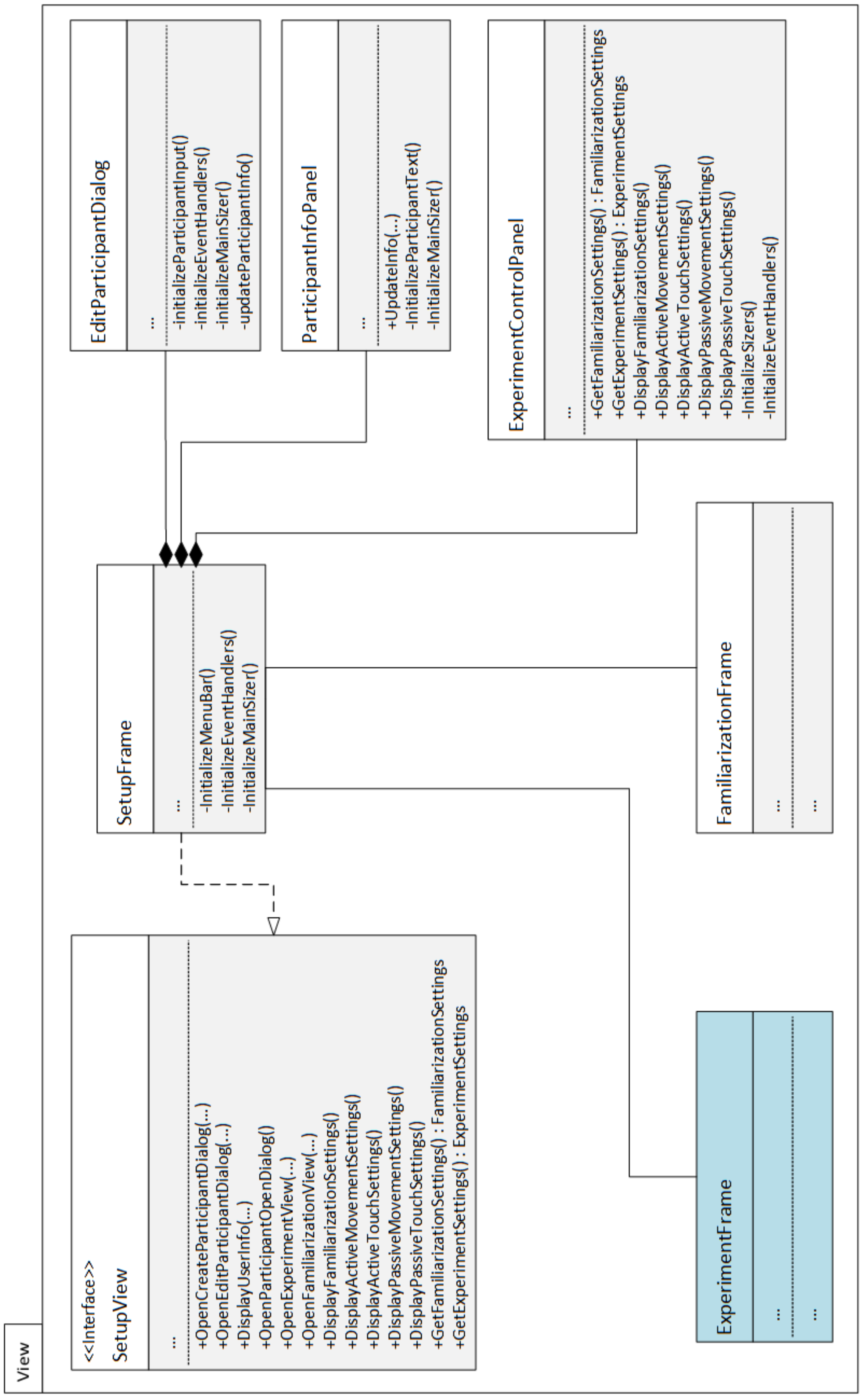
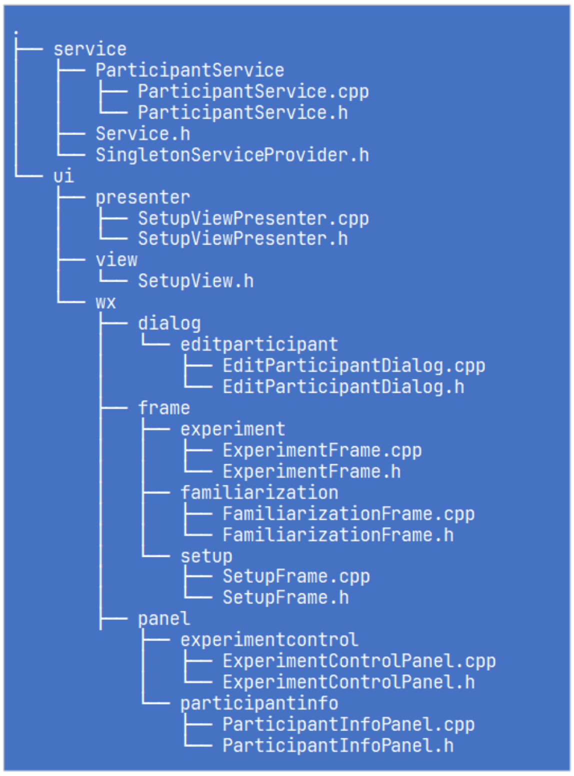
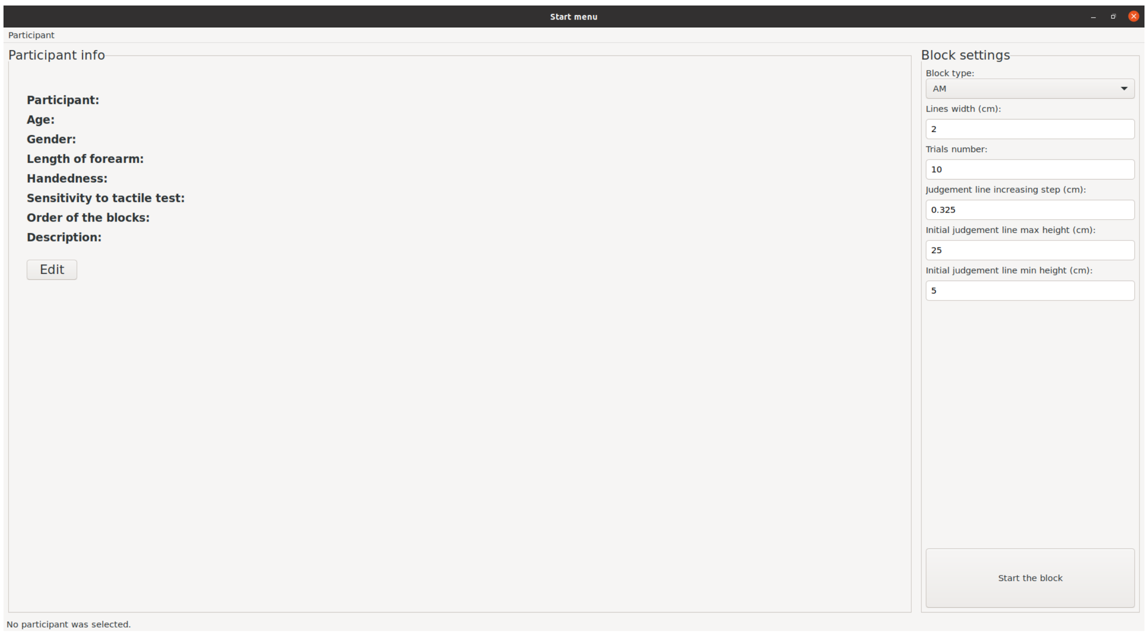

View
<<Interface>>
SetupView
...
+OpenCreateParticipantDialog(...)
+OpenEditParticipantDialog(...)
+DisplayUserInfo(...)
+OpenParticipantOpenDialog()
+OpenExperimentView(...)
+OpenFamiliarizationView(...)
+DisplayFamiliarizationSettings()
+DisplayActiveMovementSettings()
+DisplayActiveTouchSettings()
+DisplayPassiveMovementSettings()
+DisplayPassiveTouchSettings()
+GetFamiliarizationSettings() : FamiliarizationSettings
+GetExperimentSettings() : ExperimentSettings
SetupFrame
...
-InitializeMenuBar()
-InitializeEventHandlers()
-InitializeMainSizer()
EditParticipantDialog
...
-initializeParticipantInput()
-initializeEventHandlers()
-initializeMainSizer()
-updateParticipantInfo()
ParticipantInfoPanel
...
+UpdateInfo(...)
-InitializeParticipantText()
-InitializeMainSizer()
ExperimentControlPanel
...
+GetFamiliarizationSettings() : FamiliarizationSettings
+GetExperimentSettings() : ExperimentSettings
+DisplayFamiliarizationSettings()
+DisplayActiveMovementSettings()
+DisplayActiveTouchSettings()
+DisplayPassiveMovementSettings()
+DisplayPassiveTouchSettings()
-InitializeSizers()
-InitializeEventHandlers()
FamiliarizationFrame
...
...
ExperimentFrame
...
...

Figure 6.7: Graphical UI for configure experiments class diagram. Views and view components.



```
.
├── service
│   ├── ParticipantService
│   │   ├── ParticipantService.cpp
│   │   └── ParticipantService.h
│   ├── Service.h
│   └── SingletonServiceProvider.h
└── ui
    ├── presenter
    │   ├── SetupViewPresenter.cpp
    │   └── SetupViewPresenter.h
    ├── view
    │   └── SetupView.h
    └── wx
        ├── dialog
        │   └── editparticipant
        │       ├── EditParticipantDialog.cpp
        │       └── EditParticipantDialog.h
        ├── frame
        │   ├── experiment
        │   │   ├── ExperimentFrame.cpp
        │   │   └── ExperimentFrame.h
        │   ├── familiarization
        │   │   ├── FamiliarizationFrame.cpp
        │   │   └── FamiliarizationFrame.h
        │   └── setup
        │       ├── SetupFrame.cpp
        │       └── SetupFrame.h
        ├── panel
            ├── experimentcontrol
            │   ├── ExperimentControlPanel.cpp
            │   └── ExperimentControlPanel.h
            └── participantinfo
                ├── ParticipantInfoPanel.cpp
                └── ParticipantInfoPanel.h
```

Figure 6.8: Graphical UI for configure experiments directory structure.



Start menu

Participant

Participant info

Participant:
Age:
Gender:
Length of forearm:
Handedness:
Sensitivity to tactile test:
Order of the blocks:
Description:

Edit

Block settings

Block type: AM

Lines width (cm): 2

Trials number: 10

Judgement line increasing step (cm): 0.325

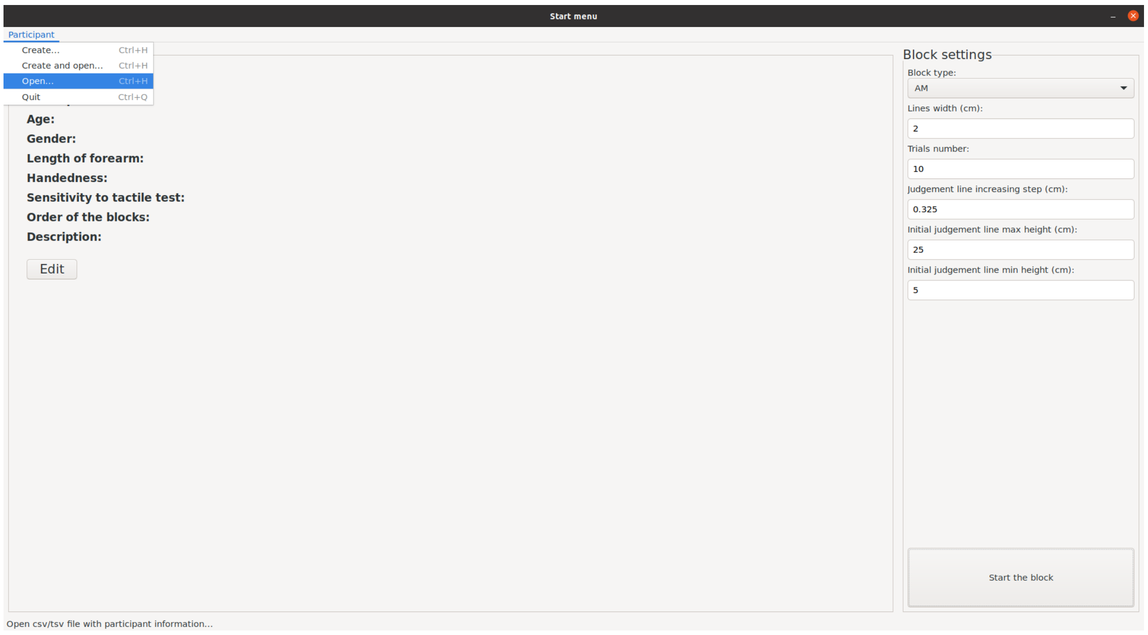
Initial judgement line max height (cm): 25

Initial judgement line min height (cm): 5

Start the block

No participant was selected.

Figure 6.9: Graphical UI for configuring. The setup screen without participant information. The active movement block selected.

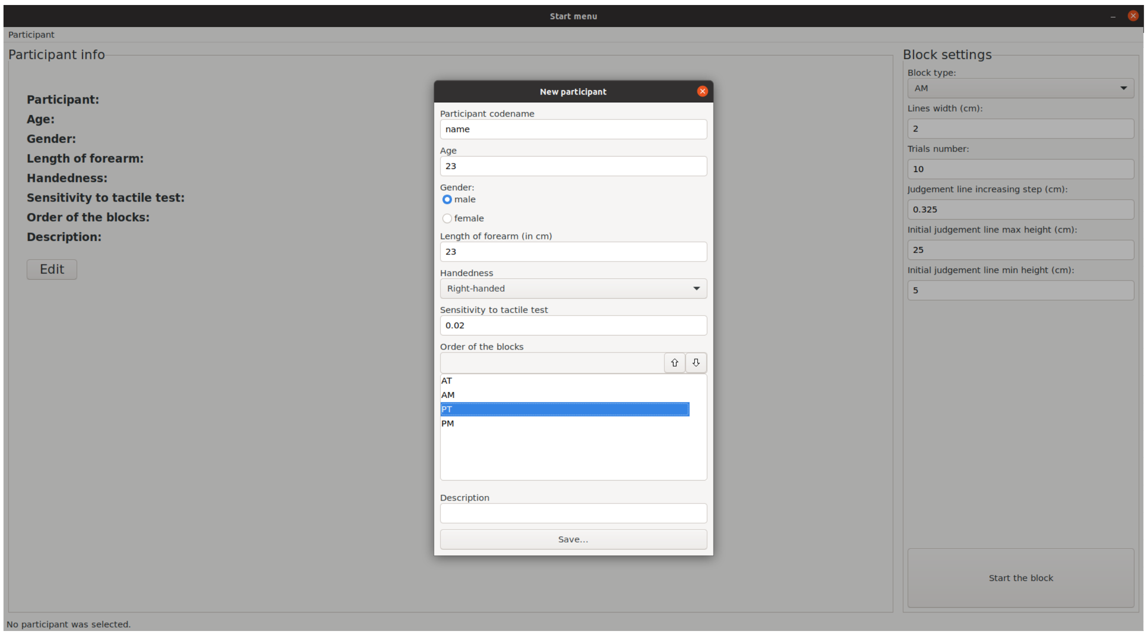


Age:
Gender:
Length of forearm:
Handedness:
Sensitivity to tactile test:
Order of the blocks:
Description:

Block settings
Block type: AM
Lines width (cm): 2
Trials number: 10
Judgement line increasing step (cm): 0.325
Initial judgement line max height (cm): 25
Initial judgement line min height (cm): 5

Open csv/tsv file with participant information...

Figure 6.10: Graphical UI for configuring. The setup screen with open “participant” menu bar and without participant information. The active movement block selected.



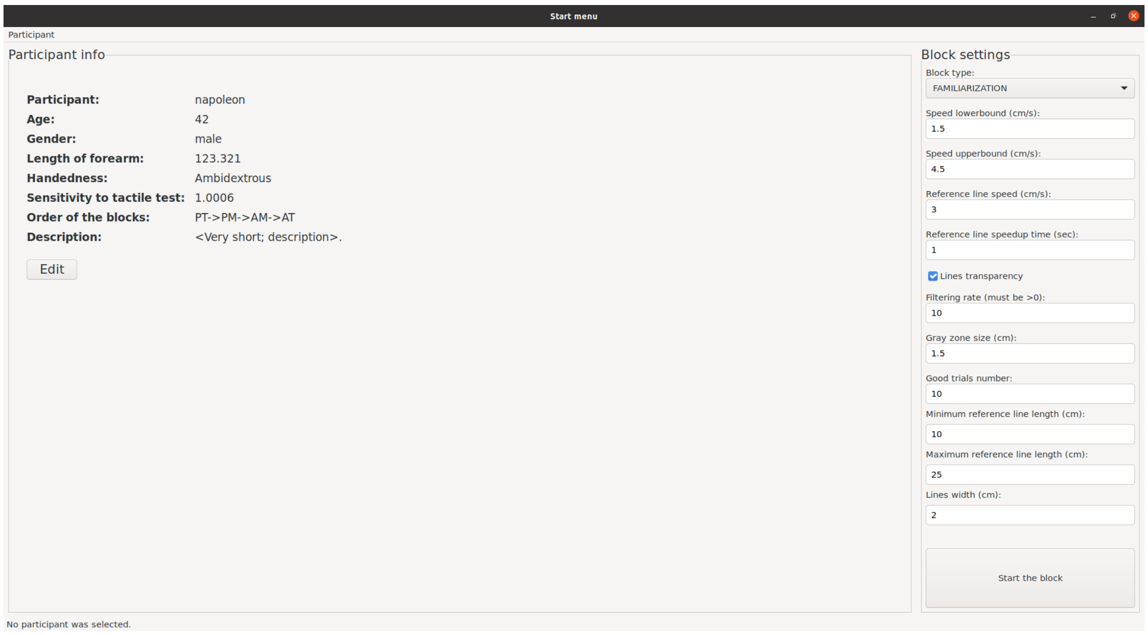
Participant info
Participant:
Age:
Gender:
Length of forearm:
Handedness:
Sensitivity to tactile test:
Order of the blocks:
Description:

Participant codename: name
Age: 23
Gender: [✓ male] female
Length of forearm (in cm): 23
Handedness: Right-handed
Sensitivity to tactile test: 0.02
Order of the blocks: AT, AM, PT, PM
Description:

Block settings
Block type: AM
Lines width (cm): 2
Trials number: 10
Judgement line increasing step (cm): 0.325
Initial judgement line max height (cm): 25
Initial judgement line min height (cm): 5

No participant was selected.

Figure 6.11: Graphical UI for configuring. The edit participant dialog. The active movement block selected.



Participant info

Participant:	napoleon
Age:	42
Gender:	male
Length of forearm:	123.321
Handedness:	Ambidextrous
Sensitivity to tactile test:	1.0006
Order of the blocks:	PT->PM->AM->AT
Description:	<Very short; description>.

Block settings

Block type: FAMILIARIZATION
Speed lowerbound (cm/s): 1.5
Speed upperbound (cm/s): 4.5
Reference line speed (cm/s): 3
Reference line speedup time (sec): 1
☑ Lines transparency
Filtering rate (must be >0): 10
Gray zone size (cm): 1.5
Good trials number: 10
Minimum reference line length (cm): 10
Maximum reference line length (cm): 25
Lines width (cm): 2

No participant was selected.

Figure 6.12: Graphical UI for configuring. The setup screen with participant information. The familiarization block selected.

6.3 Data logging - implementation

The data logging logic is implemented in the application's `FamiliarizationLoggerService` and `ExperimentLoggerService`, which are used by `FamiliarizationPresenter` and `ExperimentPresenter`, respectively. These services use methods for creating log files, implemented in the `FamiliarizationBlockInfoFile` and `ExperimentBlockInfoFile` classes, which can create files with the structure defined in Section 5.4. To make the file creation logic available for any classes that need it, the following common classes were created:

- GenericFile — the application utility that contains methods for managing files without any specific structure.
- DSVFile — the application utility that inherits from `GenericFile` and contains methods for managing DSV files (files with the delimiter-separated values format).
- HeadersAndValuesFile – the application utility that inherits from `DSVFile` and contains methods for managing DSV files with two rows that represent certain defined headers and values corresponding to those headers.

The implemented classes are shown as UML class diagrams in Figure 6.13. The directory structure for these classes is represented in Figure 6.14. Examples of every output files can be found in the "./cpp/output_data_example" directory on the project's GitLab [11] or in the attachment.

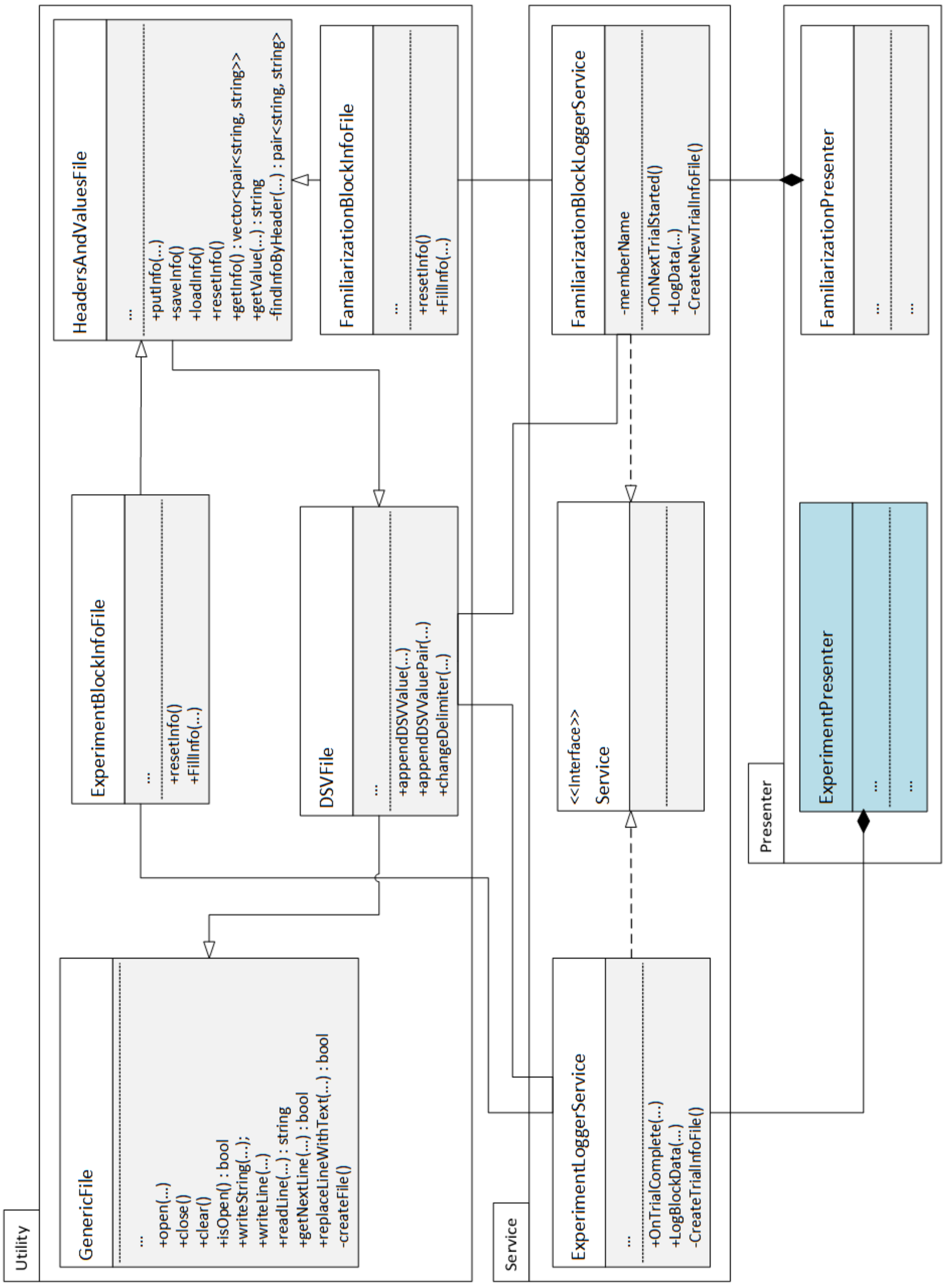
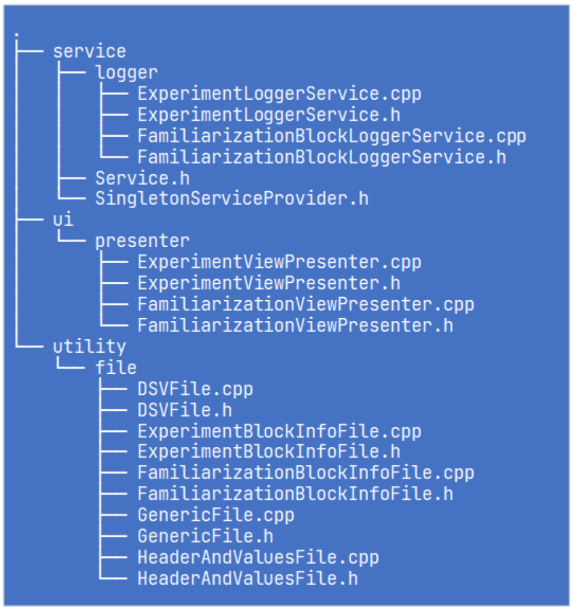


Figure 6.13: Data logging UML class diagram.



```
.
├── service
│   ├── logger
│   │   ├── ExperimentLoggerService.cpp
│   │   ├── ExperimentLoggerService.h
│   │   ├── FamiliarizationBlockLoggerService.cpp
│   │   └── FamiliarizationBlockLoggerService.h
│   ├── Service.h
│   └── SingletonServiceProvider.h
├── ui
│   └── presenter
│       ├── ExperimentViewPresenter.cpp
│       ├── ExperimentViewPresenter.h
│       ├── FamiliarizationViewPresenter.cpp
│       └── FamiliarizationViewPresenter.h
└── utility
    └── file
        ├── DSVFile.cpp
        ├── DSVFile.h
        ├── ExperimentBlockInfoFile.cpp
        ├── ExperimentBlockInfoFile.h
        ├── FamiliarizationBlockInfoFile.cpp
        ├── FamiliarizationBlockInfoFile.h
        ├── GenericFile.cpp
        ├── GenericFile.h
        ├── HeaderAndValuesFile.cpp
        └── HeaderAndValuesFile.h
```

Figure 6.14: Data logging classes directory structure.

6.4 2D extension - implementation

To implement 2D logic, I created the "compute_2D_position" method in the RobotWorker class to compute the 2D position of the robot during the hand tracking process, and the "position_2d_speed_is_under_limits" method to check if the robot's velocity and position are within limits. These methods are based on legacy application methods for computing robot position during 1D hand tracking, and all safety checks remain the same. However, the requirements of future experiments that will use this 2D extension are not known. Therefore, if any safety checks fail during the computation of the robot's position in the `RobotWorker` class, the application stops hand tracking and displays an error scene on the screen, making robot movement even safer during 2D hand tracking. Despite this, the robot's 2D movement was tested remotely in an environment where it could not collide with any objects. Risk assessment will be required once all details of a future experiment that will use 2D tracking are known.

For easy control of starting and stopping robot 2D tracking, `Start2DHandFollowing` and `Stop2DHandFollowing` methods were implemented in the `RobotService` class. Presenters and views were also created to implement scenes described in Section 5.5. Graphical representations of these scenes are shown in Figure 6.16. The UML class diagram with classes related to the 2D extension is shown in Figure 6.15. The directory structure containing these classes is represented in Figure 6.17.

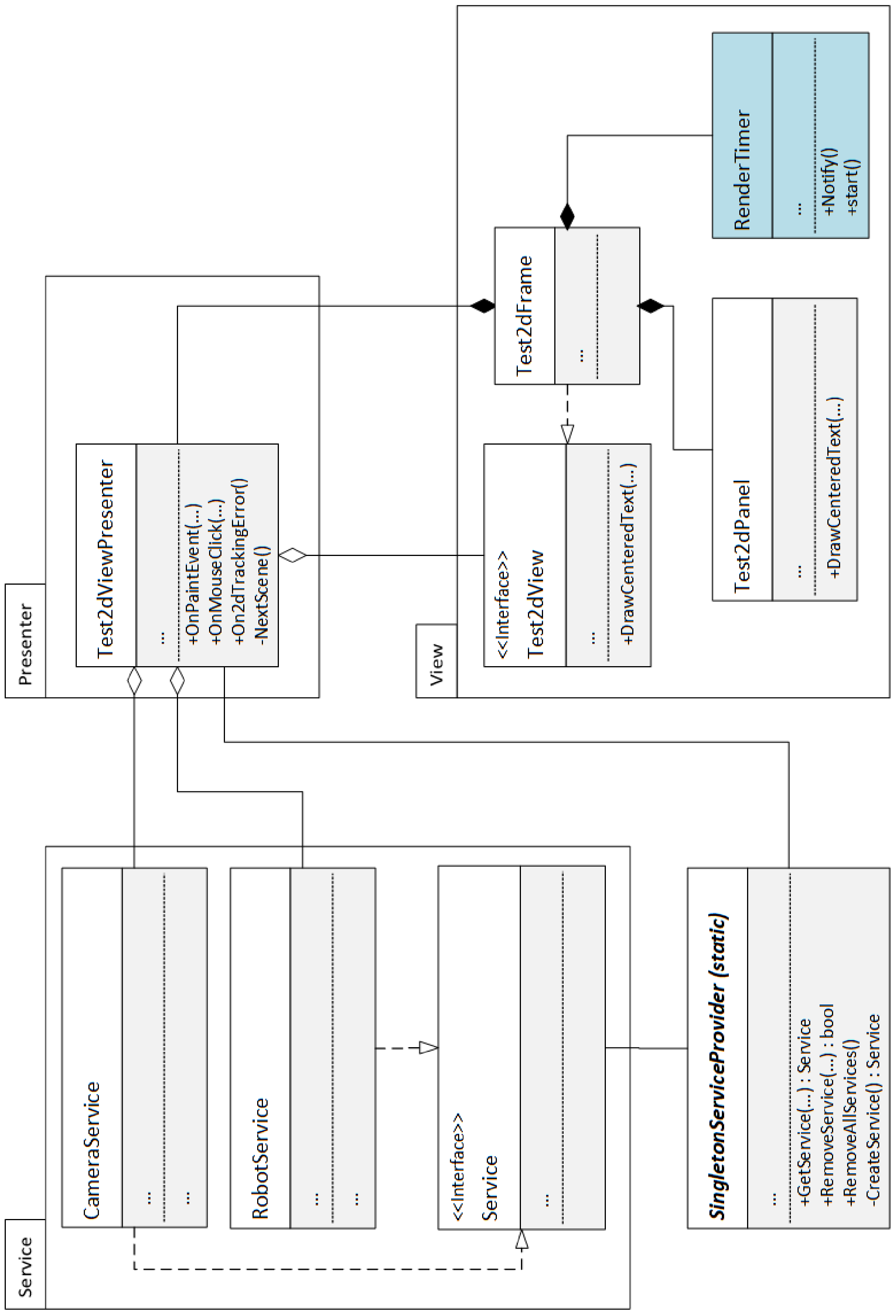
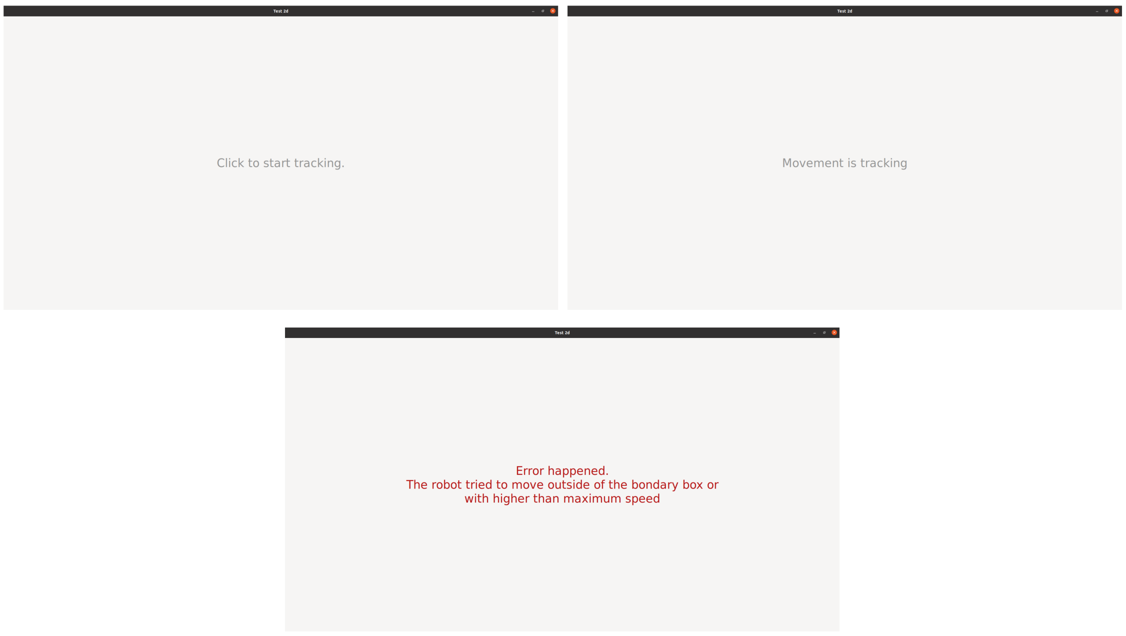
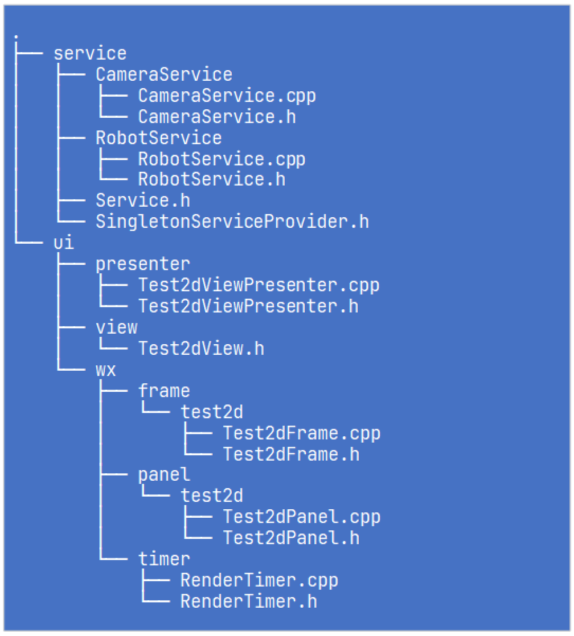


Figure 6.15: 2D extension class diagram.



Error happened.
The robot tried to move outside of the bondary box or
with higher than maximum speed

Figure 6.16: Familiarization block scenes showcase.



```
.
├── service
│   ├── CameraService
│   │   ├── CameraService.cpp
│   │   └── CameraService.h
│   ├── RobotService
│   │   ├── RobotService.cpp
│   │   └── RobotService.h
│   ├── Service.h
│   └── SingletonServiceProvider.h
└── ui
    ├── presenter
    │   ├── Test2dViewPresenter.cpp
    │   └── Test2dViewPresenter.h
    ├── view
    │   └── Test2dView.h
    └── wx
        ├── frame
        │   └── test2d
        │       ├── Test2dFrame.cpp
        │       └── Test2dFrame.h
        ├── panel
        │   └── test2d
        │       ├── Test2dPanel.cpp
        │       └── Test2dPanel.h
        └── timer
            ├── RenderTimer.cpp
            └── RenderTimer.h
```

Figure 6.17: 2D extension classes directory structure.

6.5 Unit-tests

The most critical parts of the program are methods that check and limit the robot's speed and position during the hand tracking period and generate robot trajectories when it must move from the initial position near the participant hand towards participant's arm in order to touch it and away from the arm towards the initial position near the participant hand.

After the refactoring process during the application architecture implementation, these methods were moved to the `PositionComputer` class. These methods are:

- generate_linear_trajectory: This method receives the trajectory start timestamp, current timestamp, trajectory duration, trajectory start position, and trajectory end position. Based on these values, it computes the position on the linear trajectory, which is later checked and sent to the robot.
- compute_intergoal_position: This method receives moving windows, the goal position, the time when the last robot position was received, a boolean parameter that indicates if slow mode is enabled, and current application settings that affect the robot's position and speed computations. It returns the goal position if speed and boundary checks are passed; otherwise, it enables slow mode and returns an "intergoal" position. The "intergoal" position must always be inside a defined boundary box and will cause speed limitation. This method is used during the hand tracking period.

Even though the logic of these methods wasn't changed during this work, it may be changed later and safety tests must be performed again. This means that after any change affecting the critical parts' logic, a developer must set up Qualisys cameras, turn on the robot, remotely simulate several trials without any living objects inside the robot operation zone, empirically ensure that the robot behaves as expected, and compute the robot's speed and position during those trials to ensure that they are within defined limits. To reduce the time for all of these tests, I wrote several unit tests that automatically check the correctness of the trajectory generation logic and emulate risky situations during the hand tracking period.

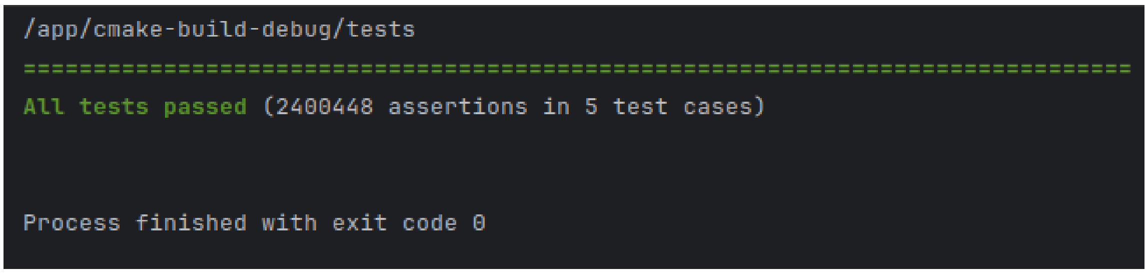
For simplicity of writing unit tests there were a number of unit-test frameworks developed such as: Google Test [22], Boost.Test [23], CppUnit [24], Cute [25] and Catch2 [26]. I chose the Catch2 framework to use in the application because of its simplicity compared with others: it is easy to install it and integrate into the project using CMake, and it provides only one main assertion macro and one main test macro, which are easy to use.

The class that contains unit-tests is called `CriticalPartsTests` and is located at the cpp/app/test directory at the project GitLab [11]. The `ProjectSettings` utility is located in the same directory and was created to dynamically retrieve current application settings from the setting.ini file for using them during tests. I wrote following unit-tests:

- Trajectory boundaries test for the `generate_linear_trajectory` method: Generates the linear trajectory from the point [-1, 5.5, 3.5] to [10, -1, 3.4] and checks that the trajectory points are always inside the boundaries defined in the application settings. The mentioned coordinates for the start and end trajectory points were chosen a priori.
- Trajectory start and end position test for the `generate_linear_trajectory` method: Generates the linear trajectory from the point [-1, 5.5, 3.5] to [10, -1, 3.4] and checks that the trajectory starts and ends at the defined points. The mentioned coordinates for the start and end trajectory points were chosen a priori.

- Trajectory traversing speed test for the `generate_linear_trajectory` method: Emulates situations where an object is traversing a generated trajectory and checks if the object speed is always constant and consistent, indicating that the trajectory is generated correctly.
- Boundary box test for the `compute_intergoal_position` method: Emulates movements inside and outside the boundary box defined in the application settings and sends them to the `compute_intergoal_position` method. Checks if `compute_intergoal_position` always returns a position inside the boundary box.
- Slow mode test for the `compute_intergoal_position` method: Emulates extremely fast movement and sends it to the `compute_intergoal_position` method. Checks if `compute_intergoal_position` always returns a position that results in a speed not exceeding 1 km/h (27.7778 cm/s).

No errors or unexpected behavior were found during the execution of these unit tests. All tests were successfully passed, and the result of the unit-tests run is shown in Figure 6.18.



```
/app/cmake-build-debug/tests
===============================================================================
All tests passed (2400448 assertions in 5 test cases)


Process finished with exit code 0
```

Figure 6.18: Example of the unit-tests run.

6.6 Project documentation

Project documentation is an important part of every application because it provides descriptions of the project implementation details for developers. This diploma thesis serves as comprehensive documentation for the logic implemented during the completion of assignments. It provides motivation for implementing different parts of the application logic in the Chapter 4, detailed logic specifications in the form of informal descriptions and a set of functional requirements in Chapter 5, and descriptions of every important implemented class in Chapter 6.

Although this should be sufficient for future developers to easily navigate and understand the application, the readme file on the project's GitLab [11] has also been updated. In addition to instructions for compiling and launching the application, the readme now contains a description of the main project directories, input and output files, a brief introduction to the new project architecture with a description of the main components, a description of the implemented unit tests, and links to the main libraries and frameworks used in the project. The combination of the information provided in this thesis and the readme file in the project's GitLab offers comprehensive documentation to further support the project.

Chapter 7

Risks assessment

The legacy application has several safety mechanisms that limit a robot's position and speed. The most important of them are described below:

- Dimension restriction: the robot follows only in one direction during the hand tracking period of experiment and familiarization trials, and other directions are fixed.
- 3D boundary box: limits the robot's operating space.
- Slow mode with reduced robot speed: activates if the robot would exceed the speed limit.
- Inner Kinova robot safety checks: shown in tables 70 and 71 in the Kinova manual [8].

None of these safety assurance methods were changed during the implementation of the new architecture and features. The experiment setup remains essentially the same, and therefore the risks in Adam Rojík's risk assessment table must also remain unchanged. However during the analysis of Adam Rojík's work it appears that no data was collected to precisely analyze the robot speed and position in the case of two of the more probable risks to confirm their negligible severity value. Those risks are:

- Incorrect position from camera input (e.g., mismatched points when some are hidden).
- Fast-moving end-effector, unexpected movement by the participant.

Therefore, I conducted tests to check if the robot position and speed will be within safe limits in case those risks appear.

20 trials were performed remotely without any living objects in the robot operating zone in a form of experiment AT block with random gains values and with a motion capture system that was calibrated as worst as possible without any efforts to reduce residuals of the cameras. The robot initial position for the trials was [0.12, -0.28, 0.18]; the boundary box was defined by following parameters: x_min=.07, x_max=.50, y_min=-.36, y_max=-.24, z_min=.16, z_max=.5, as during a normal experiment in order to restrict the robot's operating zone so that in any scenario it can only touch the participant's right arm and nothing else.

In the first 10 trials, I simulated situations where the motion capture system couldn't detect the required marker for movement repetition. To do this, I randomly hid the marker after the trial started, moved my hand, and then revealed the marker again. For the next 10 trials, I moved my hand as quickly as possible to test the slow mode. In each trial, I also attempted to move my hand in different dimensions to test the boundary box and dimension restrictions.

To evaluate the results, data from the motion capture system were used, and the speed of markers on the robot and human hand was computed and analyzed. Merged data with the human hand marker and the robot marker speed, and the motion capture system frames corresponding to these values are presented in Figures 7.1 and 7.2 for the first 10 trials and the last 10 trials respectively.

From the data, it can be seen that the maximum speed of the robot marker was 0.966 km/h and the maximum speed of the human hand marker was above 3 km/h. This indicates that the robot's speed did not exceed 1 km/h, a speed considered safe as brush bristles moving at 1 km/h is considered as safe. When the robot tried to reach a speed near 1 km/h, the application switched to slow mode with a reduced velocity of approximately 0.18 km/h.

Even when the human hand marker had a much higher speed, the robot's speed did not exceed 1 km/h, and from this it can be concluded that the robot's speed is indeed limited by a value computed from the settings parameters. Although there is no exact parameter in the settings that clearly defines the maximum speed of the robot, making it challenging to adjust safety parameters if needed. The robot speed limitation in the legacy safety assurance methods is always based on the following parameters defined in the settings.ini file:

- norm_cartesian_vel_max: Velocity limit (in m/s), computed from a moving window average (the moving window is described in [3] on page 30).
- max_vel_compensation_multiplier: The compensator for delays and inaccuracies. The max_distance parameter is based on this, as norm_cartesian_vel_max * max_vel_compensation_multiplier * time = max_distance, where time is the time difference between now and when the joint angles were received to calculate the position of the end-effector [3].
- max_dist_recovery_multiplier: A constant in the range [0, 1]. If the robot goes outside the max_distance, slow mode is activated. This is the threshold (max_dist * this) to revert back to normal mode.
- max_vel_decrease_multiplier: A constant in the range [0, 1]. This is the speed multiplier in slow mode (max_dist * this).

Clearly defining the robot's maximum velocity as a setting parameter can provide flexibility for safety assurance parameters, but editing this mechanism goes beyond the scope of the current work.

The boundary box mechanism and dimension restrictions were also tested by analyzing the robot marker position data from the 20 trials described below. The graphs representing the XY robot marker position during all of these trials are shown in Figure 7.3. Empirically, the robot's position on the Z-axis did not change during any of the test trials. It can be concluded that the robot did not exceed the boundary box limits during any of these trials, and the Y position (with the robot as the origin of a Euclidean space) during hand tracking was always approximately -66mm that is considered as safe.

In addition to the 20 conducted test trials, numerous other test trials were conducted during application improvements. After multiple tests, no other risks were registered.

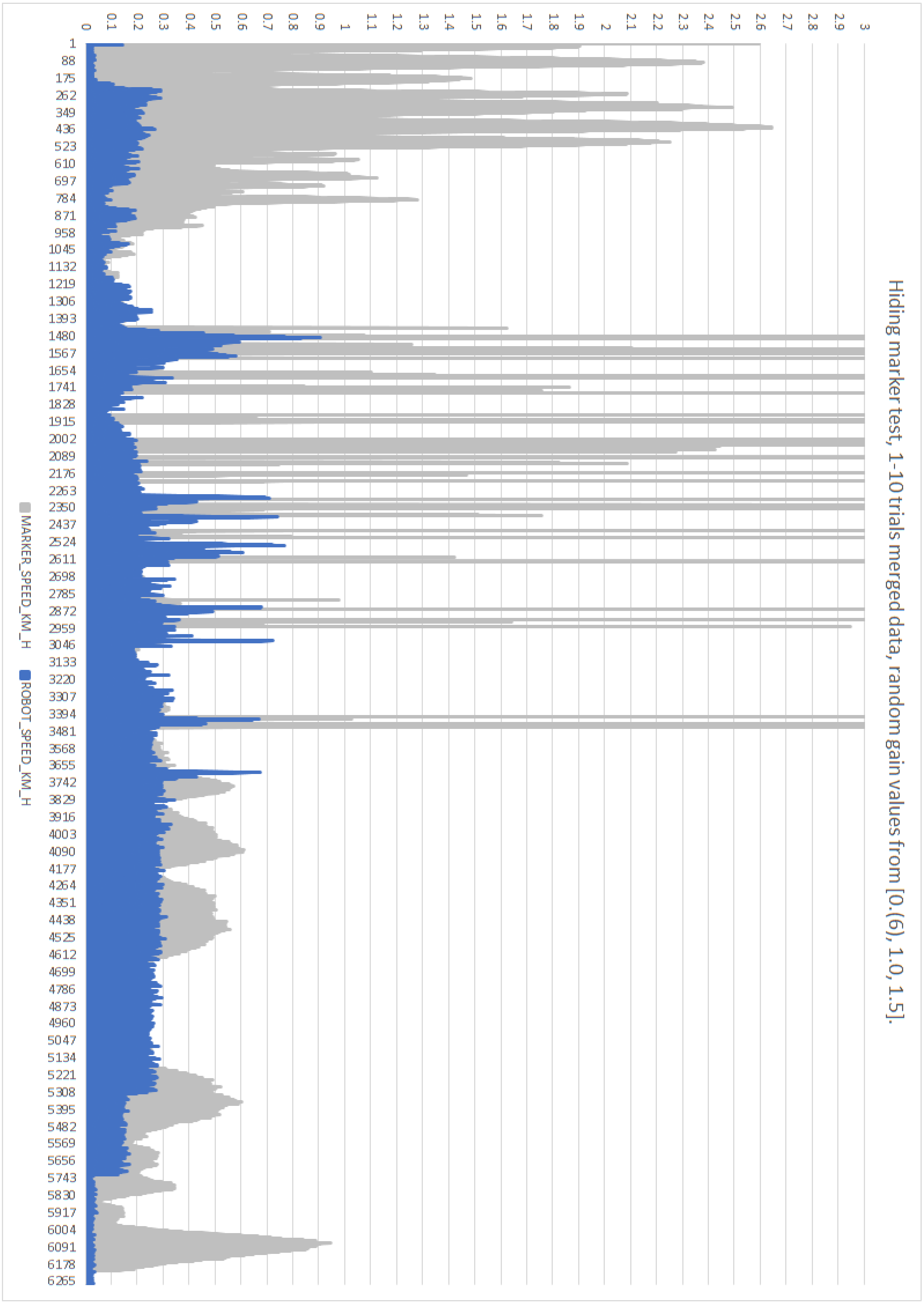


Figure 7.1: Hiding marker test results.

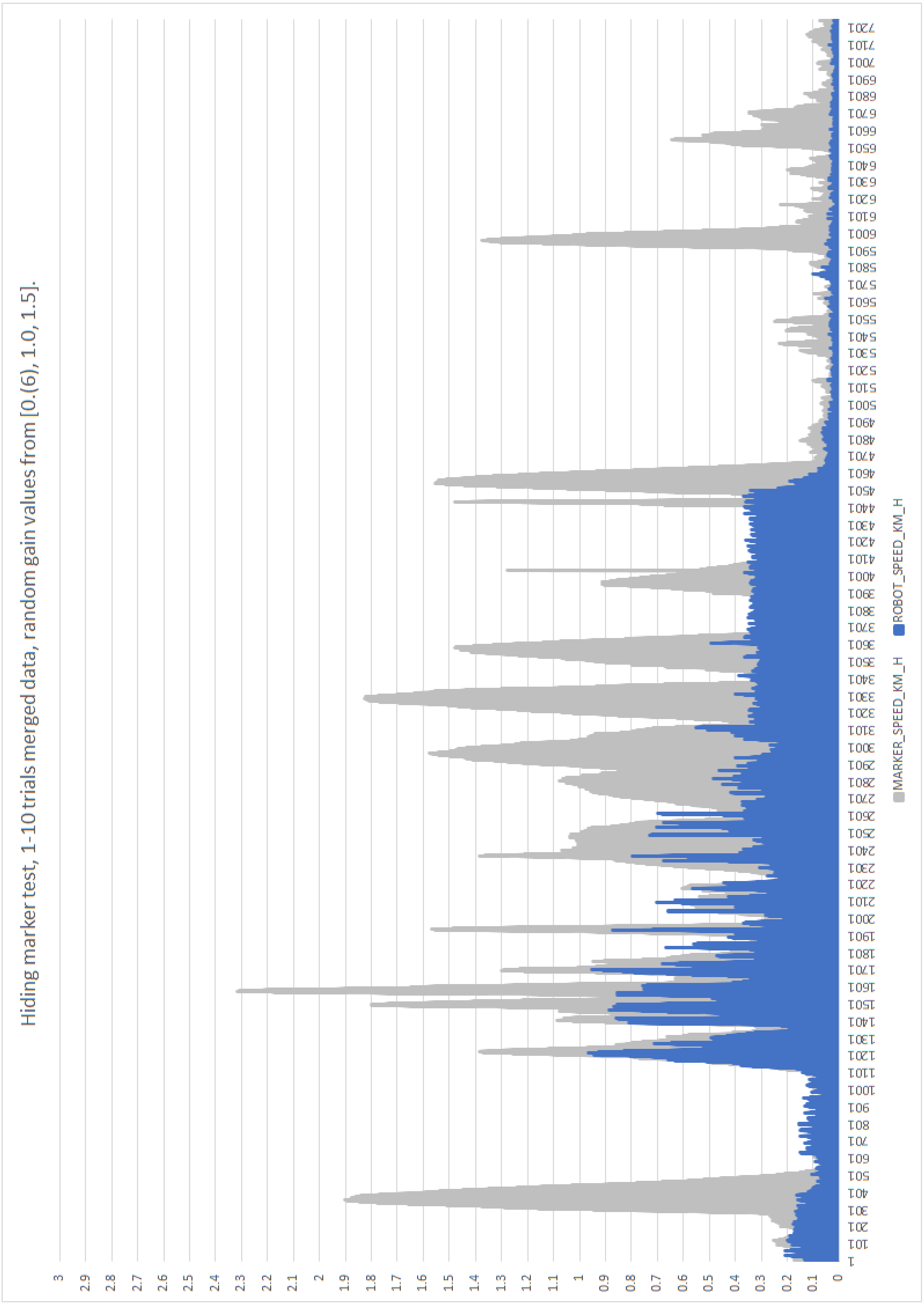


Figure 7.2: Fast movements test results.

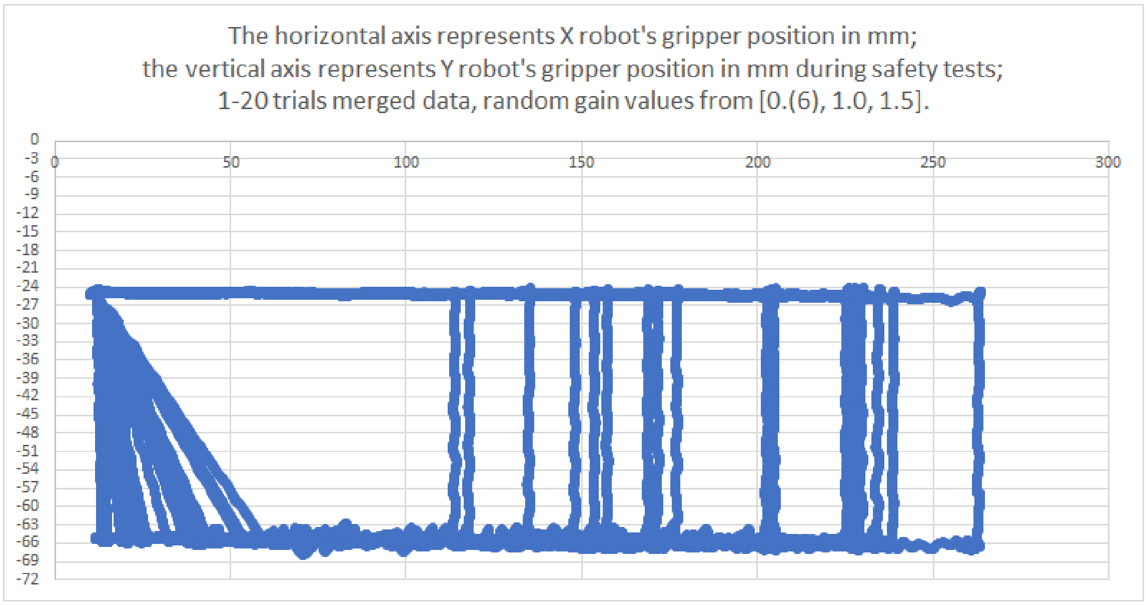


Figure 7.3: Robot's gripper positions during safety tests.

Based on the information above, it can be concluded that the risk assessment described in Table 4.2 of Adam Rojík's work [3] remains the same, and experiments with real participants may be performed.

Chapter 8

Conclusion

As a result of this thesis, the experiment investigating the interplay of tactile and motor information in constructing spatial perception has been conducted, and as the result data has been obtained for further analysis. Additionally, the tele-touch application has been fully developed and now capable of conducting experiments utilizing the concept of tele-touch.

The application is essentially a platform with a clean architecture that already provides the ability to track the hand movements and replicate it in 2D by using the robot. With minor modifications, it will be possible to use this application to conduct other similar experiments.

8.1 Meeting the Objectives

This section contains a summary of what was accomplished in accordance with the thesis assignment.

“Familiarize yourself with the Kinova Gen3 robotic platform, Qualisys motion tracking system, psychological experiment procedures [CAT22], and previous implementation [ROJ23]”: this task was fulfilled. I have read the Kinova® Gen3 Ultra lightweight robot user guide (version r9.1), the Qualisys track manager manual, the diploma work "Real-Time Teleoperation of a Robot Arm for Self-Contact", the scientific papers titled "Sensorimotor Signals Underlying Space Perception: An Investigation Based on Self-Touch" and "Interplay of Tactile and Motor Information in Constructing Spatial Self-Perception." The information obtained from these sources is presented in the Chapter 2, which contains a description of the Motion Capture system, the Kinova Gen 3 robotic platform, and psychological experiment procedures. The analysis of previous application implementations is provided in Chapter 4.

“Extend the existing implementation (and reimplement when necessary) by adding an interface to configure the experiments (for non-programmers), a familiarization phase for participants, and logging of the data for evaluation of the experiments. Extend the existing GUI where necessary. Rework existing software architecture and use suitable architecture and design patterns if needed.”: this task has been fulfilled. The existing implementation was extended and most of its parts were reimplemented by specifications described in the Chapter 5. The interface to configure experiments, designed for non-programmers, along with a participant familiarization phase and data logging for experiment evaluation, were implemented and presented in the Chapter 6. The legacy software architecture was fully reworked as it was described in Sections 4.1, 5.1 and results of the new architecture implementation is shown in the Chapter 6.

“Analyze critical parts of the program and cover it by Unit-tests. Pay specific attention to the safety of participants - see Risk assessment in [ROJ23]. It has to be guaranteed that the robot arm does not leave the defined workspace, exceed set velocity limits, or contact forces.”: this task was fulfilled. The critical parts of the program were analyzed in Section 6.5 and Chapter 7. Unit tests were implemented for these parts, as presented in Section 6.5. Safety assurance, ensuring that the robot arm does not leave the defined workspace, exceed set velocity limits, or contact forces is provided in the Chapter 7.

“Together with the supervisors, run experiments with participants (1D version - participants and robot moving on a line)”: this task has been fulfilled. The motivation for replicating the experiment was provided in Chapter 3, and a detailed description of the replicated experiment with five participants was presented in Chapter 3.

“Extend the setup to a plane (2D): Participant moves its arm on a plane which is mapped to the motion of the robot arm (subject to manipulations), which then touches the participant’s other arm.”: this task was partially completed. The block that allows a participant to control the robot in a 2D space was implemented and described in Section 6.4. However, no tests were conducted with a participant in the robot operating zone.

“Provide comprehensive documentation.”: this task has been completed. Specifications of the implemented logic and descriptions of the implemented classes were provided in Chapters 5 and 6. The source code with a clean architecture, along with a readme file with description of the main parts of the application, are available in the GitLab project [11].

Bibliography

[1] H. G. P. H. Antonio Cataldo, Lucile Dupin, "Sensorimotor signals underlying space perception: An investigation based on self-touch," *Neuropsychologia*, vol. 151, p. 107729, 2021.

[2] H. D.-J. H. G. P. H. Antonio Cataldo, Lucile Dupin, "Interplay of tactile and motor information in constructing spatial self-perception," *Current Biology*, vol. 32, no. 6, pp. 1301–1309, 2022.

[3] A. Rojík, "Real-time teleoperation of a robot arm for self-contact - master thesis," 2023.

[4] H. Lotze, *Metaphysic: In Three Books, Ontology, Cosmology, and Psychology.* Clarendon Press, 1884.

[5] B. M. S. Ernst, Marc O., "Humans integrate visual and haptic information in a statistically optimal fashion," *Nature*, vol. 415, pp. 429–433, 2002.

[6] A. M. Melvyn A. Goodale, "Separate visual pathways for perception and action," *Trends in Neurosciences*, vol. 15, pp. 20–25, 1992.

[7] H. C. Dijkerman and E. H. F. de Haan, "Somatosensory processing subserving perception and action: Dissociations, interactions, and integration," *Behavioral and Brain Sciences*, vol. 30, pp. 224–230, 2007.

[8] Kinova inc., *Kinova® Gen3 Ultra lightweight robot user guide r9.1*, 2023.

[9] Qualisys, *Qualisys Track Manager user manual*, 2022.

[10] Qualisys, "QTM Real-time Server Protocol Documentation," last accessed 18 May 2024. [Online]. Available: https://docs.qualisys.com/qtm-rt-protocol/

[11] O. Baryshnikov, "Real-time teleoperation of a robot arm for manipulating self-localization in human participants project - git repository," 2023-2024, last accessed 16 May 2024. [Online]. Available: https://gitlab.fel.cvut.cz/body-schema/tele-touch

[12] *Working Draft, Standard for Programming Language C++*, 2022, ch. 5, p. 13.

[13] "Working Draft, Standard for Programming Language C++," last accessed 22 May 2024. [Online]. Available: https://github.com/cplusplus/draft

[14] B. Stroustrup, *A Tour of C++*. Addison Wesley, 2014, ch. 1, p. 4.

[15] "Stoelting™ Touch Test Sensory Probes," last accessed 24 May 2024. [Online]. Available: https://www.fishersci.com/shop/products/touch-test-sensory-probes/10000250

[16] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison Wesley, 1998, ch. 8.

[17] S. Baker, T. Moore, F. Rocher, F. Sesser, and K. Tokoro, "Tree package," 1996-2018, last accessed 16 May 2024. [Online]. Available: https://packages.ubuntu.com/focal/tree

[18] Kinova inc., "Kinova® kortex™ api reference - git repository," 2019-2024, last accessed 18 May 2024. [Online]. Available: https://github.com/Kinovarobotics/kortex/releases

[19] wxWidgets, "wxDC Class Reference," last accessed 18 May 2024. [Online]. Available: https://docs.wxwidgets.org/3.2.0/classwx_d_c.html

[20] wxWidgets, "wxWidgets Cross-Platform GUI library," last accessed 18 May 2024. [Online]. Available: https://www.wxwidgets.org/

[21] Xavier Leclercq and the wxCharts contributors, "wxCharts - a library to create charts in wxWidgets applications," last accessed 18 May 2024. [Online]. Available: https://github.com/wxIshiko/wxCharts

[22] "GoogleTest," last accessed 22 May 2024. [Online]. Available: https://github.com/google/googletest

[23] G. Rozental, "Boost Test Library," last accessed 22 May 2024. [Online]. Available: https://www.boost.org/doc/libs/1_49_0/libs/test/doc/html/index.html

[24] "CppUnit - C++ port of JUnit," last accessed 22 May 2024. [Online]. Available: https://sourceforge.net/projects/cppunit/

[25] "CUTE Framework," last accessed 22 May 2024. [Online]. Available: https://www.cute-test.com/

[26] "Catch2 A modern, C++-native, test framework for unit-tests," last accessed 22 May 2024. [Online]. Available: https://github.com/catchorg/Catch2/tree/devel